BRNO UNIVERSITY OF TECHNOLOGY
VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ

FACULTY OF INFORMATION TECHNOLOGY
FAKULTA INFORMAČNÍCH TECHNOLOGIÍ

DEPARTMENT OF COMPUTER GRAPHICS AND MULTIMEDIA
ÚSTAV POČÍTAČOVÉ GRAFIKY A MULTIMÉDIÍ

ROBUST SPEAKER VERIFICATION WITH DEEP NEURAL NETWORKS

ROBUSTNÍ ROZPOZNÁVÁNÍ MLUVČÍHO POMOCÍ NEURONOVÝCH SÍTÍ

MASTER'S THESIS
DIPLOMOVÁ PRÁCE

AUTHOR **Bc. JÁN PROFANT**
AUTOR PRÁCE

SUPERVISOR **Ing. PAVEL MATĚJKA, Ph.D.**
VEDOUCÍ PRÁCE

BRNO 2019

Vysoké učení technické v Brně
Fakulta informačních technologií

Ústav počítačové grafiky a multimédií (UPGM) Akademický rok 2018/2019

Zadání diplomové práce

21835

Student:	**Profant Ján, Bc.**
Program:	Informační technologie Obor: Počítačová grafika a multimédia
Název:	**Robustní rozpoznávání mluvčího pomocí neuronových sítí**
	Robust Speaker Verification with Deep Neural Networks
Kategorie:	Zpracování řeči a přirozeného jazyka

Zadání:

1. Prostudujte statistické techniky pro modelování řeči, soustřeďte se na neuronové sítě.
2. Seznamte se s trénovací a testovací sadou pro soutěž NIST SRE 2018.
3. Seznamte se se systémem na verifikaci mluvčího v toolboxu Kaldi.
4. Porovnejte základní systém z Kaldi se základním systémem založeným na i-vektor schématu.
5. Adaptujte systém z Kaldi na podmínky soutěže NIST SRE 2018.
6. Podrobněji analyzujte alespoň dva bloky ze základního systemu. Například: VAD, příznaky, augmentace dat, topologie DNN, klasifikátor (PLDA), změny v trénovacích datech či vzorkovací frekvenci, ...

Literatura:

- X-vectors: Robust DNN embedings for speaker recognition: https://www.danielpovey.com/files/2018_icassp_xvectors.pdf
- Základní systém v Kaldi: https://david-ryan-snyder.github.io/2017/10/04/model_sre16_v2.html

Při obhajobě semestrální části projektu je požadováno:

- Body 1 až 4.

Podrobné závazné pokyny pro vypracování práce viz http://www.fit.vutbr.cz/info/szz/

Vedoucí práce:	**Matějka Pavel, Ing., Ph.D.**
Vedoucí ústavu:	Černocký Jan, doc. Dr. Ing.
Datum zadání:	1. listopadu 2018
Datum odevzdání:	22. května 2019
Datum schválení:	6. listopadu 2018

Abstract

The objective of this work is to study state-of-the-art deep neural networks based speaker verification systems called x-vectors on various conditions, such as wideband and narrowband data and to develop the system, which is robust to unseen language, specific noise or speech codec. This system takes variable length audio recording and maps it into fixed length embedding which is afterward used to represent the speaker. We compared our systems to BUT's submission to Speakers in the Wild Speaker Recognition Challenge (SITW) from 2016, which used previously popular statistical models - i-vectors. We observed, that when comparing single best systems, with recently published x-vectors we were able to obtain more than 4.38 times lower Equal Error Rate on SITW core-core condition compared to SITW submission from BUT. Moreover, we find that diarization substantially reduces error rate when there are multiple speakers for SITW core-multi condition but we could not see the same trend on NIST SRE 2018 VAST data.

Abstrakt

Tématem této práce je analýza nejmodernějších systémů pro rozpoznávání řečníka za použití neurónových sítí (nazývaných x-vektory) v rozličných podmínkách, jako jsou širokopásmové a úzkopásmové data, který je robustní vůči neviděnému jazyku, specifickému hluku nebo telefonnimu kodeku. Automatický systém mapuje zvukovou nahrávku variabilní délky do fixně dlouhého vektoru, který je následně využit jako reprezentace řečníka. V této práci jsme porovnali systémy založené na neurónových sítich s výsledkem VUT týmu v "Speakers in the Wild Speaker Recognition" Challenge (SITW), který využíval donedávna velmi populární statistický model - i-vektory. Pozorovali jsme, že s nedávno publikovanými x-vektory dosahujeme 4.38 krát nižší Equal Error Rate pro SITW core-core evaluační sadu v porovnání s výsledkem z roku 2016 od VUT v SITW soutěži. Kromě toho jsme ukázali, že diarizace v nahrávkach s více mluvčími významně snižuje chybovost systému pro SITW core-multi evaluační data, ale podobný trend jsme neviděli pro dataset NIST SRE 2018 VAST.

Keywords

speaker verification, speaker recognition,neural networks, x-vector, i-vector

Klíčová slova

verifikace mluvčího, rozpoznávání mluvčího, neurónové sítě, x-vector, i-vector

Reference

PROFANT, Ján. *Robust Speaker Verification with Deep Neural Networks.* Brno, 2019. Master's thesis. Brno University of Technology, Faculty of Information Technology. Supervisor Ing. Pavel Matějka, Ph.D.

Robust Speaker Verification with Deep Neural Networks

Declaration

Hereby I declare that this master's thesis was prepared as an original author's work under the supervision of Ing. Pavel Matějka PhD. The supplementary information was provided by Ing. Ondřej Novotný, voice activity detection labels were provided by Ing. Oldřich Pl-chot PhD and diarization labels and Agglomerative Hierarchical Clustering implementation were provided by M.Sc. Mireia Diez Sánchez. All the relevant information sources, which were used during preparation of this thesis, are properly cited and included in the list of references.

. . . .
Ján Profant
May 17, 2019

Acknowledgements

I would like to thank my supervisor Ing. Pavel Matějka PhD. for his extensive support. I would like to also thank MSc. Anna Silnova for her help with HT-PLDA implementation and also to my colleagues from Phonexia s.r.o., Mgr. Josef Slavíček and Ing. Michal Klčo. And a very special thanks to Nina, my partner in life.

Contents

Chapter 1

Introduction

Speaker verification (SV) is the task of authenticating the claimed identity of a speaker, based on speech signal and enrolled speaker record. Similarly to fingerprints, voice is a common type of biometric data, which every individual can produce and which can be captured. However, speech is a very complex signal carrying not only the desired content but also other various information, which might significantly influence automatic processing. This environment or *channel* has a great effect on the quality of such signal, which causes the degradation in performance of SV systems, which, in an ideal case, should be robust to these conditions.

Most speaker recognition systems in the recent years were based on i-vectors [13]. The standard i-vector approach consists of a universal background model (UBM), and a large projection matrix $\boldsymbol{T}$, that are learned in an unsupervised way to maximize the data likelihood. The projection maps high-dimensional statistics from the UBM into a low-dimensional representation, known as an i-vector. These embeddings might be scored using Euclidean distance, cosine distance but more common is to use backend with Probability Linear Discriminant Analysis (PLDA) [40].

Deep neural networks (DNNs) have evolved hand-in-hand with the digital era, which has brought about an explosion of data in all forms - images, video, audio, and text. DNNs are used nowadays in industrial automation, medical research, autonomous driving and in the most of electronic devices, such as mobile phones and personal computers.

DNNs most often found in speaker recognition are trained as acoustic models for automatic speech recognition and are then used to enhance phonetic modeling in the i-vector. Using deep neural networks as an end-to-end system for a topic of speaker verification shows as a very active area of research in the last years [50, 47, 49]. In this approach, time-delay neural network (TDNN) which works on frame level is used, and during training, it is trained to classify large dataset of speakers. Long-term speaker characteristics are captured in the network by a temporal pooling layer that aggregates over the input speech. Eventually, fixed-dimensional embeddings from the layer in a network after frame level are used to represent speaker utterance and these are called x-vectors. DNNs most often found in speaker recognition are trained as acoustic models for automatic speech recognition and are then used to enhance phonetic modeling in the i-vector. In recent years, i-vectors started to be replaced by x-vectors, mainly because of their better generalization and better discriminative properties. In this paper, we analyze the performance of both approaches, and we focus on using deep neural networks for speaker verification. Both i-vectors and x-vectors are theoretically described in Chapter 2.

DNN embedding performance appears to be highly scalable with the amount of training data. Recent speaker recognition evaluations were mainly focused on narrowband telephone speech, and even automatic systems for wideband conditions were trained using mostly telephone data. Data for NIST Speaker Recognition Evaluation 2018 (NIST SRE18) [1], were split into two main parts - Call My Net 2 (CMN2) and Video Annotation for Speech Technology (VAST). The CMN2 data are composed of PSTN and VOIP data collected outside North America, spoken in Tunisian Arabic. The VAST data are composed of audio extracted from YouTube. Similarly to NIST SRE18, our experiments are also split into two parts, experiments with telephone CMN2 data are described in Chapter 4 and experiments with wideband VAST data are described in Chapter 5.

Typically, the speaker recognition systems are trained on thousands of speech cuts from thousands of speakers. Assuming such a large amount of resources for every new domain of interest might be too expensive or even unrealistic. In Chapter 4 we analyze approaches for in-domain adaptation of speaker recognition systems, using either *unlabeled* or *labeled* data.

Usage of a large amount of wideband data for training arrived with large VoxCeleb datasets [35, 10]. We used wideband data to improve performance of speaker recognition systems on wideband conditions and our effort is summarized in Chapter 5.

The problem of speaker recognition for multi-speaker conversations is even more complicated since it is not clear how many speakers are in the recording and taking whole speech segment as one single embedding is inaccurate [48]. Speech data collected from many real-world environments violate single-speaker assumption and therefore benefit from speaker diarization as a preprocessing step. Speaker diarization is the process of grouping segments of speech according to the speaker and is sometimes referred to as the *who spoke when* task. Recently, both speaker recognition and diarization have advanced significantly due to the adoption of deep neural networks [14, 48]. Diarization and its influence on performance in multi-speaker recordings is shown in Section 5.3.2.

In this paper, we introduce numerous modifications to Kaldi [39] recipe [49], which was publicly released for the research community. We also summarized our effort during NIST SRE18 where one of our systems was used for final submission in wideband VAST dataset, as single best system for this condition.

[1] https://www.nist.gov/sites/default/files/documents/2018/08/17/sre18_eval_plan_2018-05-31_v6.pdf

Chapter 2

Theoretical Background

2.1 Speaker Recognition

Speaker recognition [25] is the identification of a person from characteristics of voices. No two individuals sound identical because their vocal tract shapes, larynx sizes, and other parts of their voice production organs are different. In addition to these physical differences, each speaker has a characteristic manner of speaking, including the use of a particular accent, rhythm, intonation style, pronunciation pattern, or choice of vocabulary.

There are two major applications of speaker recognition technologies and methodologies. If the speaker claims to be of a certain identity and the voice is used to verify this claim, this is called *verification* or *authentication*. Verification is often used when accessing internet banking using telephone, or entering the building, known as a *voice as my password*. On the other hand, *identification* is the task of determining an unknown speaker's identity. Identification determines identity from a known set of speakers and is used in virtual home assistants, such as Alexa [1].

Application dictates different speech modalities:

- **Text-dependent** - recognition system knows the text, that is spoken by a person, knowledge of spoken text can improve system performance.
- **Text-independent** - recognition system does not know the text spoken by person, more flexible system but a more difficult problem.

Automatic speaker verification pipeline is shown in Figure 2.1. In the first stage, we use voice activity detection (VAD) to identify speech frames, non-speech frames (silence, audio events) are dropped. After that, we extract features from these voiced frames, such as Mel Frequency Cepstral Coefficients (MFCCs). These features are then used in embedding extractor, which in our case outputs either i-vector or x-vector. This, of course, expects that only a single speaker is at enroll or test side. These speaker embeddings are then compared with the Probabilistic Linear Discriminant Analysis (PLDA) model getting log-likelihood ratio (LLR) scores.

2.2 Voice Activity Detection

Voice Activity Detection (VAD) is used in telecommunications, for example, in telephony to detect touch tones and the presence or absence of speech. Detection of speaker activity can

[1] https://developer.amazon.com/alexa

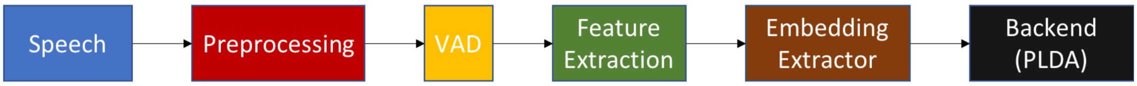

Figure 2.1: *Standard processing pipeline of state-of-the-art speaker verification system.*

be useful in responding to barge-in, for pointing to the end of an utterance in automated speech recognition, and for recognizing a word intended to trigger the start of a service, application, event, or anything else that may be deemed useful.

VAD is typically based on the amount of energy in the signal (a signal having more than a threshold level of energy is assumed to contain speech, for example) and in some cases also on the rate of zero crossings, which gives a crude estimate of its spectral content. If the signal has high-frequency components, then the zero-crossing rate is high and vice versa.

In the recent years, more advanced approach using neural networks was developed [31]. In this approach, neural network is trained to classify frames as speech or non-speech, producing per-frame scores.

2.3 Feature Extraction

Speech signal includes many features of which not all are important for speaker discrimination. An ideal feature would:

- have large between-speaker variability and small within-speaker variability
- be robust against noise and distortion
- occur frequently and naturally in speech
- be easy to measure from a speech signal
- be difficult to impersonate/mimic
- not be affected by the speaker's health or long-term variations in voice.

The number of features should also be relatively low. Traditional statistical models such as the Gaussian Mixture Model (GMM) cannot handle high-dimensional data. The number of required training samples for reliable density estimation grows exponentially with the number of features, and the computational savings are also apparent with low-dimensional features [25].

2.3.1 Mel Frequency Cepstral Coefficients

Mel Frequency Cepstral Coefficients (MFCCs) [25] are a feature widely used in automatic speech and speaker recognition. MFCCs are a representation of the short-term power spectrum of a sound, based on a linear cosine transform of a log power spectrum on a nonlinear mel scale of frequency. Figure 2.2 shows procedure, how to calculate MFCCs.

Here, we can see a more detailed description of how to calculate MFCCs according to Figure 2.2:

1. Frame the signal into short frames.

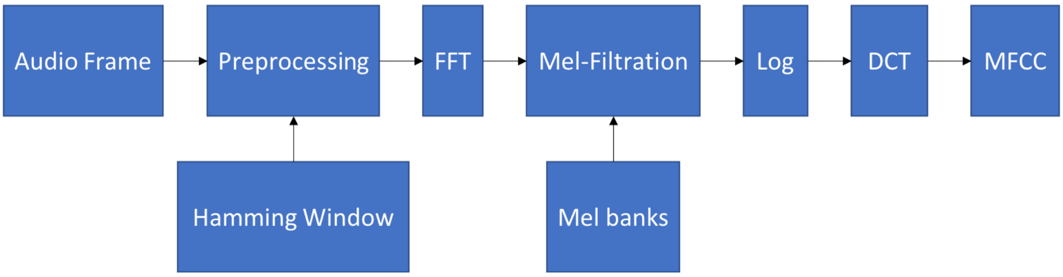

Figure 2.2: *Scheme of calculating MFCCs.*

2. For each frame calculate the periodogram estimate of the power spectrum.
3. Apply the mel filterbank to the power spectra, sum the energy in each filter.
4. Take the logarithm of all filterbank energies.
5. Take the DCT of the log filterbank energies.

2.3.2 Bottleneck Features

Bottleneck Neural-Network (BN-NN) [17, 29] refers to such topology of a NN, where one of the hidden layers has significantly lower dimensionality than the surrounding layers. A bottleneck feature vector is generally understood as a by-product of forwarding a primary input feature vector through the BN-NN and reading off the vector of values at the bottleneck layer. We have used a cascade of two such NNs for our experiments. The output of the first network is *stacked* in time, defining context-dependent input features for the second NN, hence the term Stacked Bottleneck Features. The dimensionality of the bottleneck layer was fixed to 80.

Bottleneck features were also widely used in automatic speech recognition [52, 53] and language identification [32].

2.3.3 Other Feature Sets

There are also other sets of features that might be used for speaker verification, such as perceptual linear prediction (PLP) coefficients. However, it has been observed that in general channel compensation methods are much more important than the choice of the base feature set [25].

Speakers differ not only in their voice timbre and accent/pronounciation, but also in their lexicon - the kind of words the speakers tend to use in their conversations. These feature are often reffered to as *high-level features*, where a speaker's characteristic vocabulary, the so-called *idiolect*, is used to characterize speakers [16].

2.4 i-vector

The i-vector approach has become state of the art in the speaker verification field in 2011 [13]. The approach provides an elegant way of reducing large-dimensional input data to a small-dimensional feature vector while retaining most of the relevant information. The technique was originally inspired by the Joint Factor Analysis (JFA) framework [24]. The basic principle is that on annotated data, we train the i-vector extractor and then for each speech segment, we extract the i-vector as a low-dimensional fixed-length representation of the segment. The main idea is that the speaker- and session-dependent supervectors of concatenated Gaussian Mixture Model (GMM) means, described later, can be modeled as

$$\mathbf{s} = \mathbf{m} + \mathbf{Tx}, \tag{2.1}$$

where $\mathbf{m}$ is the Universal Background Model (UBM) GMM mean supervector, $\mathbf{T}$ is a matrix of bases spanning the subspace covering the important variability (both speaker- and session-specific) in the supervector space, and $\mathbf{x}$ is a standard-normally distributed latent variable. For each observation sequence representing a segment, our i-vector $\boldsymbol{\phi}$ is the MAP point estimate of the latent variable $\mathbf{x}$ [8].

2.4.1 Gaussian Mixture Model

A Gaussian Mixture Model (GMM) is a generative model that assumes all the data points are generated from a mixture of a finite number of Gaussian (normal) distributions with unknown parameters. Gaussian distributions are important in statistics and are often used in the natural and social sciences to represent real-valued random variables whose distributions are not known. GMM is used in i-vector framework as the Universal Background Model.

The probability density of the multivariate Gaussian distribution is:

$$\mathcal{N}(\boldsymbol{x}; \boldsymbol{\mu}, \boldsymbol{\Sigma}) = \frac{1}{\sqrt{(2\pi)^P |\boldsymbol{\Sigma}|}} e^{-\frac{1}{2}(\boldsymbol{x}-\boldsymbol{\mu})^T \boldsymbol{\Sigma}^{-1}(\boldsymbol{x}-\boldsymbol{\mu})}, \tag{2.2}$$

where $\boldsymbol{\mu}$ is the mean and parameter $\boldsymbol{\Sigma}$ is variance matrix with its matrix determinant $|\boldsymbol{\Sigma}|$. Frequently used methods to estimate parameters are maximum a posteriori probability (MAP) and maximum likelihood (ML). MAP estimator chooses class with highest posteriori probability from N classes:

$$arg\ max_{\omega}\ P(\omega|x) = arg\ max_{\omega}\ P(x|\omega)\ P(\omega). \tag{2.3}$$

Maximum likelihood is a method of estimating the parameters of a statistical model given data, as follows [26]:

$$\boldsymbol{\Theta}_{ML}^{class} \ = \ arg\ max_{\boldsymbol{\Theta}} \prod_{x_i \in class} p(x_i|\boldsymbol{\Theta}), \tag{2.4}$$

where $\boldsymbol{\mu}$ is estimated as

$$\boldsymbol{\mu} = \frac{1}{T} \sum\nolimits_i \boldsymbol{x}_i, \tag{2.5}$$

and covariance matrix $\boldsymbol{\Sigma}$ as

$$\boldsymbol{\Sigma} = \frac{1}{T} \sum\nolimits_i (\boldsymbol{x}_i - \boldsymbol{\mu})(\boldsymbol{x}_i - \boldsymbol{\mu})^T. \tag{2.6}$$

GMM is then a probabilistic generative model,

$$p(\boldsymbol{x}|\Theta) = \sum\nolimits_c \mathcal{N}(\boldsymbol{x}; \boldsymbol{\mu_c}, \boldsymbol{\Sigma_c}) P_c \tag{2.7}$$

where $\boldsymbol{\Theta} = \{\boldsymbol{P}_c, \boldsymbol{\mu}_c, \boldsymbol{\Sigma}_c\}$ is set of parameters and $\sum_c \boldsymbol{P}_c = \mathbf{1}$.

A GMM is used in speaker recognition applications as a generic probabilistic model for multivariate densities capable of representing arbitrary densities, which makes it well suited for unconstrained text-independent applications [41].

2.5 Neural Networks

The term *neural network* has its origins in attempts to find mathematical representations of information processing in biological systems [54]. It has been used very broadly to cover a wide range of different models and problems.

In the field of the speech recognition, the deep neural networks (DNN)-hidden Markov model (HMM) has been shown to significantly improve speech recognition performance over the conventional Gaussian mixture model (GMM)-HMM [22]. In language identification and speaker verification, BN-NN were widely used, compressing phonetic information and improving results for both, language identification and speaker verification, as described in Section 2.3.2. Eventually, using end-to-end systems based on neural networks have risen for speech recognition [21] and also for language identification [45] and speaker verification [49].

The most successful and probably the simplest model of this type in the context of pattern recognition is the feed-forward neural network.

The linear models for regression and classification are based on linear combinations of fixed nonlinear basis functions $\phi_j(\boldsymbol{x})$ and take the form

$$y(\boldsymbol{x}, \boldsymbol{w}) = f\left(\sum_{j=1}^{M} w_j \phi_j(\boldsymbol{x})\right) \tag{2.8}$$

where $\boldsymbol{x}$ is input vector, $\boldsymbol{w}$ is set of weights and $f(\cdot)$ is a nonlinear activation function in the case of classification and is the identity in the case of regression.

Neural networks use basis functions that follow the same form as in Equation 2.8 so that each basis function is itself a nonlinear function of a linear combination of the inputs, where the coefficients in the linear combination are adaptive parameters.

This leads to the basic neural network model, which can be described as a series of functional transformations. First we construct M linear combinations of the input variables $x_1, x_2, ..., x_D$ in the form

$$a_j = \sum_{i=1}^{D} w_{ji}^{(1)} x_i + w_{j0}^{(1)} \tag{2.9}$$

where $j = 1, ..., M$ and the superscript (1) indicates that the corresponding parameters are in the first layer of the network. We refer to the parameters $w_{ji}^{(1)}$ as weights and the parameters $w_{j0}^{(1)}$ as biases. The quantities a_j are known as activations. Each of them is then transformed using a differentiable, nonlinear activation function $h(\cdot)$ to give $z_j = h(a_j)$.

These quantities correspond to the outputs of the basis functions in that, in the context of neural networks, are called hidden units. The nonlinear functions $h(\cdot)$ were generally chosen to be sigmoidal functions such as the logistic sigmoid or the hyperbolic tangent, or more commonly these days, rectified linear units [12]. The choice of activation function is determined by the nature of the data and the assumed distribution of target variables.

For standard regression problems, the activation function is the identity so that $y_k = a_k$. Similarly, for multiple binary classification problems, each output unit activation is transformed using a logistic sigmoid function so that $y_k = \sigma(a_k)$, where $\sigma(a) = \frac{1}{1+exp(-a)}$. Finally, for multiclass problems, a softmax activation function is used.

We can combine these various stages to give the overall network function that, for sigmoidal output unit activation functions, takes the form

$$y_k(\boldsymbol{x}, \boldsymbol{w}) = \sigma\left(\sum_{j=1}^{M} w_{kj}^{(2)} h\left(\sum_{i=1}^{D} w_{ji}^{(1)} x_i + w_{j0}^{(1)}\right) + w_{k0}^{(2)}\right), \tag{2.10}$$

where the set of all weight and bias parameters have been grouped together into a vector $\boldsymbol{w}$. Thus the neural network model is simply a nonlinear function from a set of input variables $\{x_i\}$ to a set of output variables $\{y_k\}$ controlled by a vector $\boldsymbol{w}$ of adjustable parameters - this process can be interpreted as a *forward propagation* of information through the network. This function can be represented in the form of a network diagram as shown in Figure 2.3.

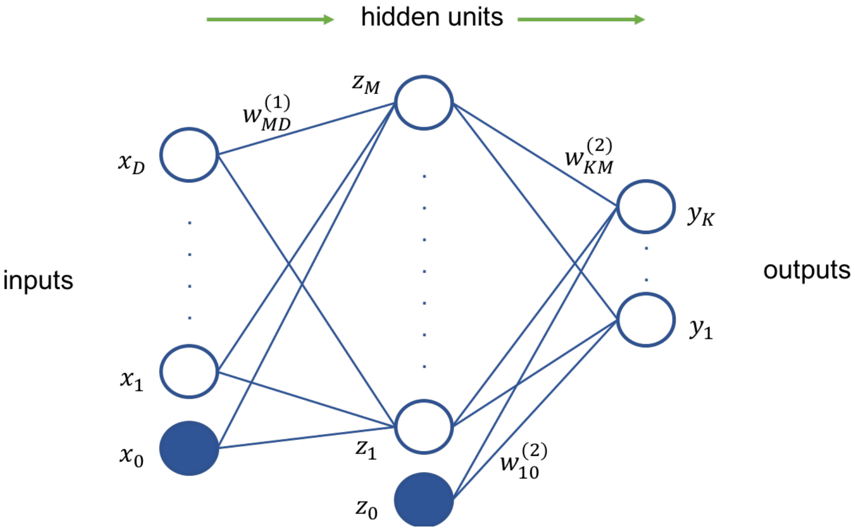

Figure 2.3: *Network diagram for the single hidden layer neural network. The input, hidden, and output variables are represented by nodes, and the weight parameters are represented by links between the nodes, in which the bias parameters are denoted by links coming from additional input and hidden variables x_0 and z_0. Arrows denote the direction of information flow through the network during forward propagation.*

If the activation functions of all the hidden units in a network are taken to be linear, then for any such network we can always find an equivalent network without hidden units. This follows from the fact that the composition of successive linear transformations is itself a linear transformation. However, if the number of hidden units is smaller than either the number of input or output units, then the transformations that the network can generate are not the most general possible linear transformations from inputs to outputs because information is lost in the dimensionality reduction at the hidden units.

Neural networks are said to be universal approximators. For example, a two-layer network with linear outputs can uniformly approximate any continuous function on a compact input domain to arbitrary accuracy provided the network has a sufficiently large number of hidden units. This result holds for a wide range of hidden unit activation functions but excluding polynomials. Although such theorems are reassuring, the critical problem is how to find suitable parameter values given a set of training data.

So far, we have viewed neural networks as a general class of nonlinear parametric functions from a vector $\boldsymbol{x}$ of input variables to a vector $\boldsymbol{y}$ of output variables. Given a training set comprising a set of input vectors $\{\boldsymbol{x}_n\}$, where $n = 1, ..., N$ together with a corresponding set of target vectors $\{\boldsymbol{t}_n\}$ we minimize the error function

$$E(\boldsymbol{w}) = \frac{1}{2}\sum_{n=1}^{N} ||\boldsymbol{y}(\boldsymbol{x}_n, \boldsymbol{w}) - \boldsymbol{t}_n||^2 \,. \tag{2.11}$$

We can clearly see, that main task is in finding a weight vector $\boldsymbol{w}$ which minimizes the chosen function $E(\boldsymbol{w})$. If we make a small step in weight space from $\boldsymbol{w}$ to $\boldsymbol{w} + \delta\boldsymbol{w}$ then the change in the error function is $\delta E \simeq \delta\boldsymbol{w}^T \nabla E(\boldsymbol{w})$, where the vector $\nabla E(\boldsymbol{w})$ points in the direction of greatest rate of increase of the error function. Because the error $E(\boldsymbol{w})$ is a smooth continuous function of $\boldsymbol{w}$, its smallest value will occur at a point in weight space such that the gradient of the error function vanishes, so that $\nabla E(\boldsymbol{w}) = 0$ as otherwise, we could make a small step in the direction of $-\nabla E(\boldsymbol{w})$ and thereby further reduce the error. Points at which the gradient vanishes are called stationary points, and may be further classified into minima, maxima, and saddle points.

Our goal is to find a vector $\boldsymbol{w}$ such that $E(\boldsymbol{w})$ takes its smallest value. However, the error function typically has a highly nonlinear dependence on the weights and bias parameters, and so there will be many points in weight space at which the gradient vanishes or is numerically very small. Furthermore, there will typically be multiple inequivalent stationary points and in particular multiple inequivalent minima. A minimum that corresponds to the smallest value of the error function for any weight vector is said to be a global minimum. Any other minima corresponding to higher values of the error function are said to be local minima. For a successful application of neural networks, it may not be necessary to find the global minimum (and in general it will not be known whether the global minimum has been found), but it may be necessary to compare several local minima in order to find a sufficiently good solution.

Because there is no hope of finding an analytical solution to the equation $\nabla E(\boldsymbol{w}) = 0$ we resort to iterative numerical procedures. The optimization of continuous nonlinear functions is a widely studied problem, and there exists an extensive literature on how to solve it efficiently. Most techniques involve choosing some initial value $\boldsymbol{w}^{\mathbf{0}}$ for the weight vector and then moving through weight space in a succession of steps of the form

$$\boldsymbol{w}^{\tau+1} = \boldsymbol{w}^{\tau} + \nabla\boldsymbol{w}^{\tau} \tag{2.12}$$

where τ labels the iteration step. Different algorithms involve different choices for the weight vector update $\nabla\boldsymbol{w}^{\tau}$. Many algorithms make use of gradient information and therefore require that, after each update, the value of $\nabla E(\boldsymbol{w})$ is evaluated at the new weight vector $\nabla\boldsymbol{w}^{\tau+1}$.

The simplest approach to using gradient information is to choose the weight update to comprise a small step in the direction of the negative gradient, so that

$$\boldsymbol{w}^{\tau+1} = \boldsymbol{w}^{\tau} - \eta\nabla\boldsymbol{w}^{\tau} \tag{2.13}$$

where the parameter $\eta > 0$ is known as the *learning rate*. After each such update, the gradient is re-evaluated for the new weight vector and the process repeated. Recently, technique know as *stochastic gradient descent*, makes an update to the weight vector based on one data point or randomly generated subset of points at a time. This update is repeated by cycling through the data either in sequence or by selecting points at random with replacement [5].

Next goal might be to find an efficient technique for evaluating the gradient of an error function $E(\boldsymbol{w})$ for a feed-forward neural network. This can be achieved using a local message passing scheme in which information is sent alternately forwards and backwards through the network and is known as error backpropagation. Consider first a simple linear model in which the outputs y_k are linear combinations of the input variables x_i so that

$$y_k = \sum_i w_{ki} x_i \tag{2.14}$$

together with an error function that, for a particular input pattern n, takes the form

$$E_n = \frac{1}{2}\sum_k (y_{nk} - t_{nk})^2 \tag{2.15}$$

where $y_{nk} = y_k(\boldsymbol{x}_n, \boldsymbol{w})$. The gradient of this error function with respect to a weight w_{ij} is given by

$$\frac{\partial E_n}{\partial w_{ji}} = (y_{nj} - t_{nj})x_{ni} \tag{2.16}$$

which can be interpreted as a local computation involving the product of an error signal $y_{nj} - t_{nj}$ associated with the output end of the link w_{ji} and the variable x_{ni} associated with the input end of the link.

In a general feed-forward network, each unit computes a weighted sum of its inputs of the form

$$a_j = \sum_i w_{ji} z_i \tag{2.17}$$

where as we could see in 2.9. Consider the evaluation of the derivative of E_n with respect to a weight w_{ji}. The outputs of the various units will depend on the particular input pattern n. However, in order to keep the notation uncluttered, we shall omit the subscript n from the network variables. First we note that E_n depends on the weight w_{ji} only via the summed input a_j to unit j. We can therefore apply the chain rule for partial derivatives to give

$$\frac{\partial E_n}{\partial w_{ji}} = \frac{\partial E_n}{\partial a_j}\frac{\partial a_j}{\partial w_{ji}}. \tag{2.18}$$

We will use a notation

$$\delta \equiv \frac{\partial E_n}{\partial a_j} \tag{2.19}$$

where the δ's are often referred to as *errors*. Using 2.17 we can write

$$\frac{\partial a_j}{\partial w_{ji}} = z_i. \tag{2.20}$$

Substituting 2.19 and 2.20 into 2.18 we obtain

$$\frac{\partial E_n}{\partial w_{ji}} = \delta_j z_i \tag{2.21}$$

which means, that the required derivative is obtained simply by multiplying the value of δ for the unit at the output end of the weight by the value of z for the unit at the input end of the weight (where $z = 1$ in the case of a bias), so in order to evaluate the derivatives, we need only to calculate the value of δ_j for each hidden and output unit in the network and then apply 2.21.

For the output units, we therefore use

$$\delta_k = y_k - t_k. \tag{2.22}$$

To evaluate the δ's for hidden units, we again make use of the chain rule for partial derivatives,

$$\delta_j \equiv \frac{\partial E_n}{\partial a_j} \equiv \sum_k \frac{\partial E_n}{\partial a_k}\frac{\partial a_k}{\partial a_j}, \tag{2.23}$$

where the sum runs over all units k to which unit j sends connections. The arrangement of units and weights is illustrated in Figure 2.4. Note that the units labelled k could include other hidden units and/or output units. In writing down 2.23, we are making use of the fact that variations in a_j give rise to variations in the error function only through variations in the variables a_k. If we now substitute the definition of δ given by 2.19 into 2.23, and make use of 2.17, we obtain the following *backpropagation* formula

$$\delta_j = h^{'}(a_j) \sum_k w_{kj} \delta_k, \tag{2.24}$$

which tells us that the value of δ for a particular hidden unit can be obtained by propagating the δ's backwards from units higher up in the network, as illustrated in Figure 2.4. Because we already know the values of the δ's for the output units, it follows that by recursively applying 2.24 we can evaluate the δ's for all of the hidden units in a feed-forward network, regardless of its topology.

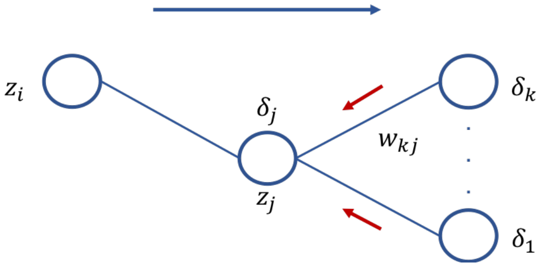

Figure 2.4: *Illustration of the calculation of δ_j for hidden unit j by backpropagation of the δ's from those units k to which unit j sends connections. The blue arrow denotes the direction of information flow during forward propagation, and the red arrows indicate the backward propagation of error information.*

2.5.1 Time-Delay Neural Networks

Time-Delay Neural Network (TDNN) is a type of architecture of a neural network that has been used quite successfully in a number of practical applications, especially in speech [37, 46]. A TDNN is similar to a multi-layer neural network in that all connections feed forward. The difference is that with the TDNN, the inputs to any node can consist of the outputs of nodes not only during the current time step t, but during some number d of previous/future time steps $(t + d, ..., t + 2, t + 1, t, t - 1, t - 2, ..., t - d$. Visualization of the TDNN layer is shown in Figure 2.5.

The activation function for node i at time t in such a network is given by

$$y_i^t = h(\sum_{j=1}^{i-1} \sum_{k=0}^{d} y_j^{t-k} w_{ijk}) \tag{2.25}$$

where y_i^t is the output of node i at time t, w_{ijk} is the connection strength to node i from the output of node j at time $t - k$, and h is the activation function [11].

Time-Delay layers are equal to one dimensional convolution layers used in many popular frameworks, such as PyTorch [36] or TensorFlow [1].

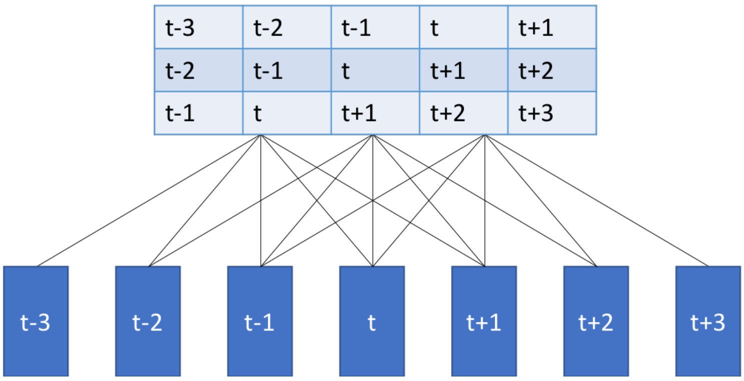

Figure 2.5: *Time-Delay layer in neural network. Blue rectangles are sorted in time and concatenated in time. This refers to one dimensional convolution used in many frameworks.*

2.6 x-vector

Using deep neural networks (DNN) to capture speaker characteristics is currently a very active research area. The used system is a feed-forward DNN that computes speaker embeddings from variable-length acoustic segments [47, 49, 50]. The network consists of layers that operate on speech frames, a statistics pooling layer that aggregates over the frame-level representations, additional layers that operate at the segment-level, and finally, a soft-max output layer, all layers with their respective contexts are shown in Table 2.1 and diagram of x-vector architecture is shown in Figure 2.6. The nonlinearities are rectified linear units (ReLUs).

Suppose there are K speakers in N training segments. Then $P(spkr_k|x_{1:T}^{(n)})$ is the probability of speaker k given T input frames $x_1^{(n)}, x_2^{(n)}, ..., x_T^{(n)}$. The quantity d_{nk} is 1 if the speaker label for segment n is k, otherwise it is 0. The network is then trained to classify training speakers using a multi-class cross entropy objective function

$$E = -\sum_{n=1}^{N}\sum_{k=1}^{K} d_{nk} ln(P(spkr_k|x_{1:T}^{(n)})) \tag{2.26}$$

Ultimately, the goal of training the network is to produce embeddings that generalize well to speakers that have not been seen in the training data. Therefore, any layer after the statistics pooling layer is a sensible place to extract the embedding from.

2.6.1 E-TDNN x-vector

The extended version of the TDNN described in Section 2.6, which is the default architecture in public Kaldi recipes is described here. Table 2.2 summarizes the extended network (E-TDNN) architecture. The two main differences are a slightly wider temporal context of the TDNN (due to the addition of *layer 7*), and interleaving dense layers in between the convolutional layers (equivalent to the 1x1 convolutions used in computer vision architectures). The network outputs posterior probabilities for the training speakers, and it was trained by minimizing a categorical cross-entropy. The x-vector is extracted from layer 12 prior to the ReLU non-linearity.

Table 2.1: *The embedding DNN architecture. x-vectors are extracted at layer segment6, before the nonlinearity. The statistics pooling layer receives the output of the final frame-level layer as input, aggregates over the input segment, and computes its mean and standard deviation. This segment-level statistics are concatenated together and passed to two additional hidden layers and finally, the soft-max output layer [49].*

Layer	Layer context	Total context
frame1	[t-2,t+2]	5
frame2	{t-2,t,t+2}	9
frame3	{t-3,t,t+3}	15
frame4	{t}	15
frame5	{t}	15
stats pooling	[0, T]	T
segment6	{0}	T
segment7	{0}	T
softmax	{0}	T

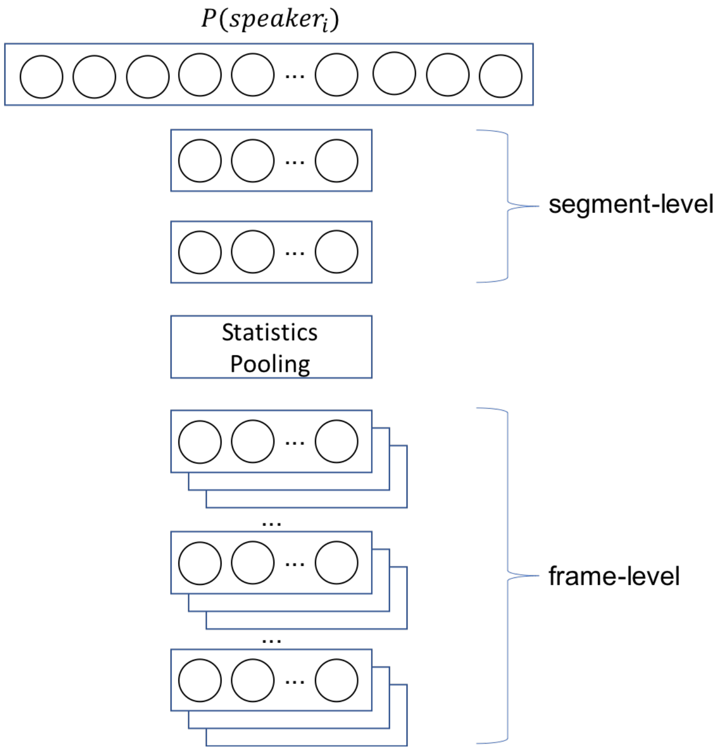

Figure 2.6: *Diagram of the DNN. Segment-level embeddings can be extracted from any layer of the network after the statistics pooling layer.*

2.7 Backend

As shown in Section 3.4, the full automatic speaker recognition pipeline can be divided into two main parts, the first one is the embedding extraction and the second one is backend.

Table 2.2: *Extended TDNN x-vector architecture.*

Layer	Layer Type	Layer context	Size
1	TDNN-ReLU	[t-2,t+2]	512
2	Dense-ReLU	t	512
3	TDNN-ReLU	{t-2, t, t+2}	512
4	Dense-ReLU	t	512
5	TDNN-ReLU	{t-3, t, t+3}	512
6	Dense-ReLU	t	512
7	TDNN-ReLU	{t-4, t, t+4}	512
8	Dense-ReLU	t	512
9	Dense-ReLU	t	512
10	Dense-ReLU	t	1500
11	Pooling (mean + stddev)	Full-seq	2x1500
12	Dense(Embedding)-ReLU		512
13	Dense-ReLU		512
14	Dense-SoftMax		512

Backend models use speaker embeddings to provide representative results. In this Section, we described frequently used backend techniques.

2.7.1 Linear Discriminant Analysis

Linear Discriminant Analysis (LDA) is most commonly used as a dimensionality reduction technique in the pre-processing step for pattern-classification and machine learning applications. The goal is to project a dataset onto a lower-dimensional space with good class-separability in order to avoid overfitting and also reduce computational costs. In Figure 2.7 we can see, to which direction we want to transfer our data.

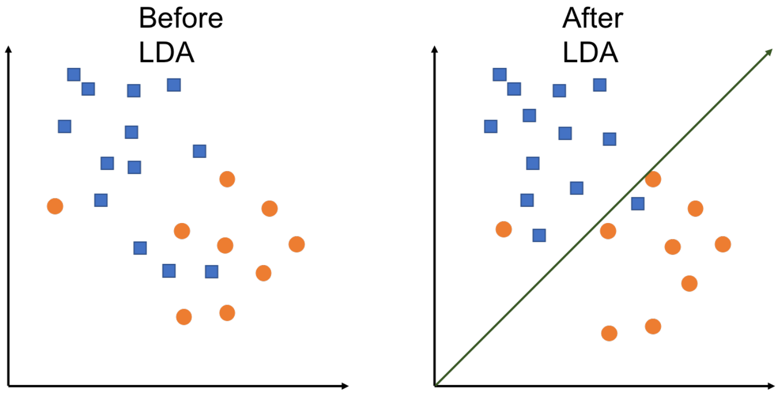

Figure 2.7: *LDA: Maximize distance between $\boldsymbol{\mu_c}$ of classes and minimize average variance of classes.*

Let us recall that LDA is based on computing the between-class and within-class covariance matrices Σ_B and Σ_W respectively, whose Maximum-Likelihood (ML) is given as

$$\boldsymbol{\Sigma_B} = \frac{1}{N}\sum_{\boldsymbol{c=1}}^{\boldsymbol{C}} \boldsymbol{N_c}(\boldsymbol{\mu_c} - \boldsymbol{\mu})(\boldsymbol{\mu_c} - \boldsymbol{\mu})^T \tag{2.27}$$

$$\boldsymbol{\Sigma_W} = \frac{1}{N}\sum_{\boldsymbol{c=1}}^{\boldsymbol{C}}\sum_{\boldsymbol{n=1}}^{\boldsymbol{N_c}} (\boldsymbol{\phi_{n,c}} - \boldsymbol{\mu})(\boldsymbol{\phi_{n,c}} - \boldsymbol{\mu})^T, \tag{2.28}$$

where $\boldsymbol{\phi_{n,c}}$, the n-th data point in class c, C is number of classes, N_c is number of datapoints in class c, μ_c is the mean of the data belonging to class c:

$$\boldsymbol{\mu_c} = \frac{1}{N_c}\sum_{\boldsymbol{n=1}}^{\boldsymbol{N_c}} \boldsymbol{\phi_{n,c}}, \tag{2.29}$$

where $\boldsymbol{\phi_{n,c}}$, the n-th data point in class c, and μ is the global mean of the data, computed as

$$\boldsymbol{\mu} = \frac{1}{N}\sum_{\boldsymbol{n=1}}^{\boldsymbol{N}} \boldsymbol{\phi_n}. \tag{2.30}$$

LDA emphasizes discrimination of data belonging to different classes, and it does so by solving the generalized eigenvalue problem:

$$\boldsymbol{\Sigma_B v_m} = \lambda_m \boldsymbol{\Sigma_B v_m}, \tag{2.31}$$

with $V = [\boldsymbol{v_1}, \dots , \boldsymbol{v_{\hat{M}}}]$ for $\hat{M}$ largest eigen-values λ_m, and applying V as

$$\boldsymbol{\phi_{LDA}} = \boldsymbol{V^T}\phi. \tag{2.32}$$

Class separability for each basis is often expressed by the Fisher ratio and is equal to the basis corresponding eigen-value [20].

2.7.2 Probabilistic Linear Discriminant Analysis

To facilitate comparison of i-vectors and x-vectors in a verification trial, the distribution of i-vectors and x-vectors is modeled using a Probabilistic Linear Discriminant Analysis (PLDA) model [40, 23]. First, consider only a special form of PLDA, a *two-covariance model*, in which speaker and inter-session variability are modeled using across-class and within-class full covariance matrices Σ_{ac} and Σ_{wc}. The two-covariance model is a generative linear-Gaussian model, where latent vectors $\mathbf{y}$ representing speakers (or more generally classes) are assumed to be distributed according to prior distribution

$$p(\mathbf{y}) = \mathcal{N}(\mathbf{y}; \boldsymbol{\mu}, \Sigma_{ac}). \tag{2.33}$$

For a given speaker represented by a vector $\hat{\mathbf{y}}$, the distribution of i-/x-vectors is assumed to be

$$p(\boldsymbol{\phi}|\hat{\mathbf{y}}) = \mathcal{N}(\boldsymbol{\phi}; \hat{\mathbf{y}}, \Sigma_{wc}). \tag{2.34}$$

Figure 2.8 depicts this situation.

The ML estimates of the model parameters, $\boldsymbol{\mu}$, Σ_{ac}, and Σ_{wc}, can be obtained using an EM algorithm as in [23].

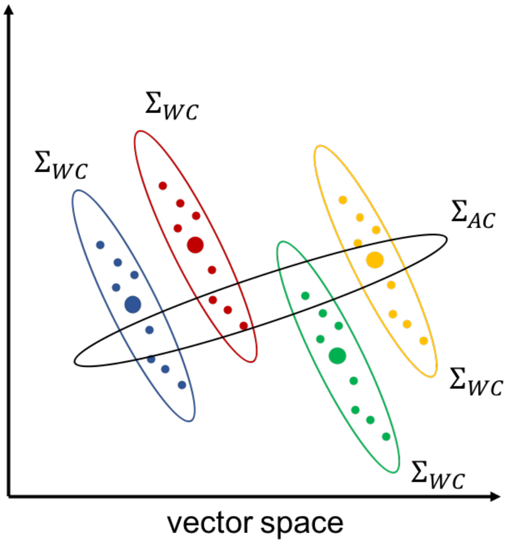

Figure 2.8: *Demonstration of PLDA: the bold points represent the speaker identities in the vector space. Provided that we know the speaker identity, the conditional distribution of the vectors is given by the within-class covariances, depicted by the ellipses around the speaker identities [19].*

Heavy-Tailed PLDA

Heavy-Tailed (HT-PLDA) model was presented in [23], where the Gaussian priors where replaced by Student's t distribution. The generative HT-PLDA model is shown in graphical model notation in Figure 2.9 and is defined as follows. For every speaker, i, let all of the available observations of that speaker (N_i of them) be denoted as $R_i = \{r_{ij}\}_{j=1}^{N_i}$, where the $r_{ij} \in \mathbb{R}^D$ are speaker embeddings (i-vectors, x-vectors) of dimension D. For every speaker, a hidden speaker identity variable, $z_i \in \mathbb{R}^d$ is drawn from the standard $d-$dimensional normal distribution. We require $d \ll D$. The heavy-tailed behaviour is obtained by drawing for every observation a hidden *precision scaling factor*, $\Lambda_{ij} > 0$, from a gamma distribution $\mathcal{G}(\alpha, \beta)$ parametrized by $\alpha = \beta = \frac{\upsilon}{2} > 0$. The parameter υ is known as the *degrees of freedom.* Finally, given the hidden variables, the observations are drawn from the multivariate normal:

$$P(r_{ij}|z_i, \lambda_{ij}) = \mathcal{N}(r_{ij}|Fz_i, (\lambda_{ij}W)^{-1}), \tag{2.35}$$

where F is the D-by-d factor loading matrix and where W is a D-by-D positive definite precision matrix. The model parameters are υ, F, W.

This model does not allow closed-form scoring or training. Either the hidden scaling factors or the hidden speaker identity variables can be integrated out in closed form, but not both. This means that we have to find approximations for both scoring and training. We make use of a new approximation, the Gaussian likelihood approximation, as recently published in [7, 44].

As observed in [28], unity length normalization of the i-vectors or x-vectors indicates that Gaussian PLDA is as effective as HT-PLDA without unity length normalization.

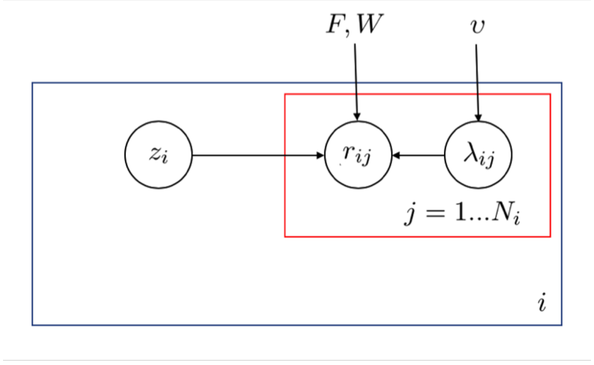

Figure 2.9: *Heavy-tailed PLDA model and its parameters.*

2.7.3 Score Normalization

For speaker verification systems, score normalization is one of the standard steps in producing well-normalized speaker verification scores [9, 30]. Without the normalization, different distributions of a target and non-target scores can be obtained for two different enrolled speaker models.

This makes it impossible to set a single detection threshold for the scores obtained from the different speaker models. Similarly, for the same speaker model, the score distributions can vary depending on the test utterance condition (recording channel, acoustic conditions or a language of the utterance) which calls for a condition-dependent threshold. Setting the threshold is also very important for production usage.

Typically, the normalization step shifts and scales the distributions for the individual models and conditions to allow for a single detection threshold. The shifts and scales are usually estimated using a set of utterances so-called normalization cohort.

Z-norm

Zero score normalization [41] employs impostor score distribution for enrollment file. It uses a cohort $\varepsilon = \{\epsilon_i\}_{i=1}^{N}$ speakers which we assume to be different from the speakers in utterances e and t. The cohort scores are

$$S_e = \{s(e, \epsilon_i))\}_{i=1}^{N} \tag{2.36}$$

and are formed by scoring enrollment utterance e with all files from cohort ε. The normalized score is then:

$$s(e,t)_{z-norm} = \frac{s(e,t) - \mu(S_e)}{\sigma(S_e)}, \tag{2.37}$$

where $\mu(S_e)$ is mean and $\sigma(S_e)$ is standard deviation of S_e.

T-norm

Test score normalization [4] is similar to Z-norm with the difference that it normalizes the impostor score distribution for the test utterance. T-norm can be expressed by:

$$S_t = \{s(t, \epsilon_i)\}_{i=1}^{N} \tag{2.38}$$

$$s(e,t)_{t-norm} = \frac{s(e,t) - \mu(S_t)}{\sigma(S_t)} \tag{2.39}$$

where $\mu(S_t)$ is mean and $\sigma(S_t)$ is standard deviation of S_t.

S-norm

The symmetric normalization (S-norm) computes an average of normalized scores from Z-norm and T-norm [23]. S-norm is symmetrical, therefore $s(e,t) = s(t,e)$, while the previously mentioned score normalization techniques depend on the order of e and t.

$$s(e,t)_{s-norm} = \frac{s(e,t)_{z-norm} + s(e,t)_{t-norm}}{2} \tag{2.40}$$

Adaptive score normalization refers to the type of score normalization when there is only a subset S_{sub} chosen from scores S. All scores S are ordered, and the N highest values are chosen and propagated into S_{sub}, where N is hyperparameter.

2.8 Diarization

In the real world scenarios, it is usually not guaranteed, that there is an exclusively single speaker in the whole audio recording. Therefore, the speaker diarization, the process of partitioning an audio stream with multiple people into homogeneous segments associated with each individual, is an important part of speaker recognition systems. By solving the problem of *who spoke when*, speaker diarization has applications in many important scenarios, such as understanding medical conversations, video captioning and more. Example of diarization output is shown in Figure 2.10.

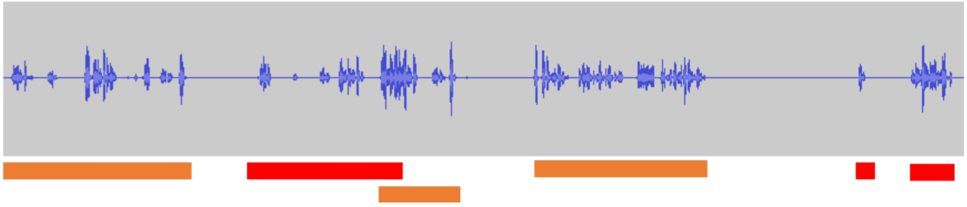

Figure 2.10: *Example output of diarization on single channel audio. Different colors in the bottom indicate different speakers.*

The speaker diarization used in this work is method based on the Bayesian Hidden Markov Model described in [14], in which states represent speaker specific distributions and transitions between states represent speaker turns. The transitions probabilities are set to favor staying in the same speakers to avoid too frequent speaker turns. As in the i-vector or JFA models, speaker distributions are modeled by GMMs with parameters constrained by eigenvoice priors to facilitate discrimination between speakers.

2.8.1 Variational Bayes

Most speaker diarization methods address the task in two steps

1. Segment input into speaker segments.

2. Run clustering algorithm on top of these segments, such as K-Means or Agglomerative Hierarchical Clustering (AHC) [51].

Approach to speaker diarization, where the sequence of speech features representing a conversation is assumed to be generated from a Bayesian Hidden Markov Model (HMM) is used. HMM states represent speakers, and the transitions between the states correspond to the speaker turns. The speaker (or HMM state) specific distributions are modeled by Gaussian Mixture Models (GMMs). In order to robustly learn the speaker specific distributions, a strong informative prior is imposed on the GMM parameters, which makes use of eigenvoices just like i-vectors or Joint Factor Analysis (JFA) [24].

2.8.2 Segmentation Based Approach

Segmentation is typically the first stage of cluster-based speaker diarization algorithms and is intended to divide the speech into short segments that are assumed to have a single or dominant speaker.

The common practice is to divide the signal into utterance segments based on VAD marks. Any long speech blocks are further subdivided to 1-2 seconds. Features or speaker's utterance representation are then extracted from these blocks, usually in the form of i-vectors or x-vectors. The segments are subsequently clustered according to these extracted features. Agglomerative Hierarchical Clustering (AHC) is one popular method because the clustering can be dictated by distance-based stopping criteria instead of assuming some number of speakers. This distance-based criterion can be used many metrics, for example, Euclidean distance, cosine distance or even PLDA scoring.

This segmentation based approach is usually used before variational Bayes described in Section 2.8.1.

2.8.3 K-Means

K-Means is well known and one of the easiest clustering methods [2]. All objects are classified as belonging to one of k groups where k is a hyperparameter. Cluster membership is determined by calculating the centroid for each group and assigning each object to the closest centroid. This approach minimizes the overall within-cluster dispersion by iterative reallocation of cluster members.

The pseudo code of K-Means clustering is described here:

1. Choose k as the number of clusters.

2. Initialize the codebook vectors of the k clusters, for example randomly.

3. For every sample vector compute the distance between the new vector and every cluster's codebook vector and choose the closest one.

4. If the assignment of the vectors and their corresponding clusters is the same as in the previous step, the algorithm has converged.

5. If not, recompute new centers of the clusters using newly assigned labels and go back to 3.

However, the k-means algorithm is susceptible to noise in data, and it negatively affects the result of the clustering.

Chapter 3

Experimental Setup

3.1 NIST SRE

The speaker recognition evaluation (SRE) is the series of speaker recognition evaluations conducted by the US National Institute of Standards and Technology (NIST) since 1996. SRE are prestige and datasets released during evaluation work as an excellent benchmark when comparing individual approaches on testing conditions.

The objectives of the evaluation series are

- to explore promising new ideas in speaker recognition
- to support the development of advanced technology incorporating these ideas
- to measure and calibrate the performance of the current state of technology

The evaluations are intended to be of interest to all researchers working on the general problem of text-independent speaker recognition.

SRE18 [1] was focusing on speaker detection over conversational telephone speech (CTS) collected outside North America. In addition to CTS recorded over a variety of handsets (PSTN), voice over IP (VOIP) data, which were also collected outside North America, as well as audio from video (AfV) were included as development and test material in SRE18.

The task for SRE18 is *speaker detection*: given a segment of speech and the target speaker enrollment data, automatically determine whether the target speaker is speaking in the segment. A segment of speech (test segment) along with the enrollment speech segment(s) from a designated target speaker constitute a *trial*. The system is required to process each trial independently and to output a log-likelihood ratio (LLR), using natural (base e) logarithm, for that trial. The LLR for a given trial including a test segment u is defined as follows:

$$LLR(u) = log\frac{P(u|H_o)}{P(u|H_1)}, \tag{3.1}$$

where $P(\cdot)$ denotes the probability distribution function (pdf), and H_0 and H_1 represent the null (i.e., u is spoken by the enrollment speaker) and alternative (i.e., u is not spoken by the enrollment speaker) hypotheses, respectively.

The LLR provides a self-contained ratio of the probability of the voices being from the same speaker versus the alternate hypothesis of them being from different speakers. That is, the LLR is meaningful on its own (i.e., a LLR of 2 means the same-speaker hypothesis

[1] https://www.nist.gov/itl/iad/mig/nist-2018-speaker-recognition-evaluation

is 100 times more likely than the alternate hypothesis). To obtain well-calibrated LLRs from a system, the system training data, including calibration training data, must closely represent the conditions of the trial being evaluated. In practice, this prerequisite is often unfulfilled due to acoustic or other differences between system-development data and the audio observed during use; a problem known as condition mismatch [9].

3.2 Data

All data we used either for training or testing purposes were data allowed by NIST for SRE18. In this section, we describe data used for training as well as for evaluation.

3.2.1 Training Data

Training data defines the amount and category of resources which are allowed to build speaker recognition system with. System training is limited to specific common data sets (with assigned LDC [2] identification) which are as follows

- 1996–2008 NIST SRE Data (LDC2009E10)
- 2010 NIST SRE and Follow-up Data (LDC2012E09)
- 2012 NIST SRE Test Set (LDC2016E45)
- 2016 NIST SRE Development Set (LDC2018E47)
- 2016 NIST SRE Test Set (LDC2018E30)
- Comprehensive Switchboard with transcripts (LDC2018E48)
- Comprehensive Fisher English with transcripts (LDC2018E49)
- MIXER6 (LDC2013S03)
- 2018 NIST SRE Development (dev) Set (LDC2018E46)
- Speakers In The Wild (SITW) [34]
- VoxCeleb1 [35] and VoxCeleb2 [10]

3.2.2 Evaluation Data

Since we are building robust speaker recognition system, we decided not to include some of the training corpora into the training set and use them for testing purposes instead, specifically 2016 NIST SRE Test Set and all testing subsets from SITW, VoxCeleb1, and VoxCeleb2. In Table 3.1 we can see datasets distribution and the corresponding number of target and non-target scores. It is important to note that *sre16EvalYUE* and all SRE18 CMN2 datasets also contains multi-session scores - 3 enroll recordings compared to single test recording. Also, *sitwEvalM-C* and SRE18 VAST enroll recordings may contain more than one speaker and thus needs to run speaker segmentation. Since *sre18DevVAST* is very small and the results may be very noisy, we will not evaluate this condition.

Since SRE18 data are split into two main domains, we decided to split our test datasets to match testing condition as much as possible:

[2] https://www.ldc.upenn.edu/

Table 3.1: *Overview of test conditions - number of files, number of speakers and number of trials.*

Condition	Files	Speakers	Target Trials	Non-Target Trials
sitwEvalC-C	1202	180	3658	718130
sitwEvalM-C	2275	180	10045	2000638
voxc1	4715	40	32276	32276
sre16EvalYUE	5449	100	19298	946098
sre18DevCMN2	1741	35	7830	100265
sre18DevVAST	37	10	27	243
sre18EvalCMN2	13451	188	60675	2002332
sre18EvalVAST	416	101	315	31500

1. Call My Net 2 (CMN2)

   (a) 2016 NIST SRE Test Set (LDC2018E30) Cantonese evaluation condition (*sre16EvalYUE*)

   (b) 2018 NIST SRE Development (dev) Set (LDC2018E46) CMN2 evaluation condition (*sre18DevCMN2*)

   (c) 2018 NIST SRE Evaluation (eval) Set CMN2 evaluation condition (*sre18EvalCMN2*)

2. Video Annotation for Speech Technology (VAST)

   (a) SITW core-core evaluation condition [34] (*sitwEvalC-C*)

   (b) SITW multi-core evaluation condition [34] (*sitwEvalM-C*)

   (c) VoxCeleb1 evaluation condition [35] (*voxc1*)

   (d) 2018 NIST SRE Development (dev) Set (LDC2018E46) VAST evaluation condition (*sre18DevVAST*)

   (e) 2018 NIST SRE Evaluation (eval) Set VAST evaluation condition (*sre18EvalVAST*)

3.3 Evaluation Metrics

Speaker recognition performance may be represented in many metrics that describe the system's behavior. Here, we present some well-known metrics that will be used later. We also described Diarization Error Rate (DER) which is used in the evaluation of diarization systems.

3.3.1 Equal Error Rate

The equal error rate can be evaluated based on False Acceptance rate (FAR) and False Rejection Rate (FRR). FAR specifies the fraction of access attempts by an unenrolled individual that are nevertheless deemed a match. FRR specifies the fraction of access attempts by a legitimately enrolled individual that is nevertheless rejected. Therefore for better accuracy, FAR and FRR must be low. The point at which FAR and FRR intersect is called Equal Error Rate (ERR). EER of any system gives system performance independent of the threshold. Therefore, lower the ERR, better the system performance [3].

3.3.2 Cost Model

A basic cost model is used to measure the speaker detection performance and is defined as a weighted sum of FAR and FRR probabilities for some decision threshold θ as follows

$$C_{Det}(\theta) = C_{Miss} \times P_{Target} \times P_{Miss}(\theta) \times + C_{FalseAlarm} \times (1 - P_{Target}) \times P_{FalseAlarm}(\theta), \quad (3.2)$$

where the parameters of the cost function are C_{Miss} (cost of a missed detection - usually equal to one), $C_{FalseAlarm}$ (cost of a spurious detection - usually equal to one) and P_{Target} (a priori probability of the specified target speaker) [42].

To improve the interpretability of the cost function C_{Det} it is normalized by $C_{Default}$ which is defined as the best cost that could be obtained without processing the input data (i.e., by either always accepting or always rejecting the segment speaker as matching the target speaker, whichever gives the lower cost), as follows

$$C_{Norm}(\theta) = \frac{C_{Det}(\theta)}{C_{Default}}, \quad (3.3)$$

where $C_{Default}$ is defined as

$$C_{Default} = min \begin{cases} C_{Miss} \times P_{Target} \\ C_{FalseAlarm} \times (1 - P_{Target}). \end{cases} \quad (3.4)$$

3.3.3 Detection Error Tradeoff

Current standard in speech verification applications to use when evaluating performance of the system for all FAR or FRR points is the Detection Error Tradeoff (DET) Curve. In the DET curve, error rates are plotted on both axes, giving uniform treatment to both types of error, and use a scale for both axes which spreads out the plot and better distinguishes different well performing systems and usually produces plots that are close to linear. For more information, see [27].

3.3.4 CLLR

In the case of speaker recognition, information theoretic measure may be computed that considers how well all scores represent the likelihood ratio and that penalizes for errors in score calibration. This performance measure is defined as

$$C_{llr} = \frac{1}{2 \times log(2)} \times (\frac{\sum log(1 + \frac{1}{s})}{N_{TT}} + \frac{\sum log(1 + s)}{N_{NT}}), \quad (3.5)$$

where the first summation is over all target trials N_{TT}, the second is over all non-target trials N_{NT}, and s represents a trial's likelihood ratio [6].

3.3.5 Diarization Error Rate

To measure the performance of a diarization system, we use the diarization error rate (DER) as our metric, which is defined by the evaluations campaigns organized by NIST. It compares the differences between the ground-truth reference segmentation and the generated diarization output. The final result is the sum of three types of errors and can be written like this:

$$DER = E_{\text{Miss}} + E_{\text{FA}} + E_{\text{Spkr}} \quad (3.6)$$

where E_{Miss} is the percentage of missed speech error (speaker not attributed when speech exists), E_{FA} is the percentage of false alarm error (speaker attributed in non-speech segment), E_{Spkr} is the percentage of speaker missclassification error (wrong speaker labelling according to reference segmentation). Lower DER indicates better diarization performance. Additionally, a non-scoring collar of 250 msec [55] is generally adopted in both sides of the ground-truth segment boundaries to eliminate the effects of inevitably inaccurate labeling.

3.4 Pipeline Setup

In this section, we describe the setup of the experiments for all individual components in the pipeline described in Section 2.1.

3.4.1 Voice Activity Detection

We used VAD that was used in previous SRE or Language Recognition Evaluations (LRE). VAD we used consists of two parts

- a neural network which produces per-frame scores and
- a postprocessing stage which builds the segments based on the scores.

The neural network was trained on the Fisher English. The input features for the NN consist of 15 log-Mel filterbank outputs and 3 Kaldi-pitch features [18]. The output of the network is then classified as speech or non-speech [31].

VAD labels were provided by Ing. Oldřich Plchot PhD.

3.4.2 Acoustic i-vector

Traditional i-vector system is similar to i-vector system in [29]. This system uses voice activity detection described in Section 3.4.1. The features are 20-dimensional MFCCs with a frame-length of 25ms that are mean normalized over a sliding window of up to 3 seconds. Delta and acceleration are appended to create 60 dimension feature vectors. The UBM is a 2048 component full-covariance GMM. This system uses a 400-dimensional i-vector extractor, LDA to 150 dimensions and gaussian PLDA for scoring.

3.4.3 Phonetic bottleneck i-vector

This i-vector system incorporates phonetic bottleneck features (BNF) described in Section 2.3.2 from an ASR DNN acoustic model and is similar to [29]. The BNFs are concatenated with the same 20 dimensional MFCCs described in Section 3.4.2 plus deltas to create 100 dimensional features. This system uses voice activity detection described in 2.2, 600 dimensional i-vector extractor, LDA to 250 dimensions and gaussian PLDA for scoring.

3.4.4 x-vector

We used original features configuration of x-vector recipe [49] obtained from [3] - 23-dimensional filterbanks with a frame-length of 25ms, mean-normalized over a sliding window of up to 3 seconds. We slightly modified our voice activity detector from Section 3.4.1 and extended all speech frames by 15 frames to the left and also to the right, effectively extending the

[3] https://david-ryan-snyder.github.io/2017/10/04/model_sre16_v2.html

amount of speech that is passed into neural network, as shown in [33]. Also, we analyzed and applied some of the possible mentioned improvements for x-vector based architecture based on [33], such as larger number of augmentation (128 000 in original recipe vs. 256 000 in our recipe) and we also used larger number of epochs (3 in original recipe compared to 6 in our recipe) and this system will be used as our baseline x-vector system. We used the same data for x-vector training as in original recipe from [49]. If not specified otherwise, we used 512-dimensional x-vector projected into 128-dimensional space using LDA. For scoring, we used gaussian PLDA backend.

3.4.5 Diarization

We used 19 MFCC+Energy coefficients (without any normalization) as features for diarization. We only ran the diarization on segments that contain speech according to our VAD. We used 1024-component, diagonal covariance GMM-UBM, and 400-dimensional i-vectors. The UBM and the total variability matrix were trained on the VoxCeleb1 and VoxCeleb2 datasets. A hierarchical agglomerative clustering (AHC) algorithm based on PLDA scores between i-vectors estimated on 200 ms segments was performed to initialize the assignment of frames to speakers for the VB algorithm [14].

Diarization labels were provided by M.Sc. Mireia Diez Sánchez.

Table 4.1: *Baseline results on telephone conditions for i-vectors and x-vectors.*

System	sre16EvalYUE EER[%]	sre18DevCMN2 EER[%]	sre18EvalCMN2 EER[%]
i-vector	13.05	17.43	19.17
BN i-vector	11.51	16.62	17.75
x-vector	5.91	10.41	11.36
E-TDNN	5.35	9.27	9.72

Chapter 4

Experiments - CMN2 condition

In this chapter we analyze the performance of our systems on telephone conditions, mainly Cantonese subset of NIST SRE 2016 evaluation set and recent NIST SRE 2018 Call My Network2 (CMN2) data.

4.1 Baseline Systems

First, we ran our baseline systems for i-vectors and x-vectors as described in Section 3.4. Results for telephone conditions are shown in Table 4.1. We can see, that BN i-vector systems outperform acoustic i-vector system in all our datasets. All i-vector system are however greatly outperformed by our x-vector baseline - EER for similar systems were reduced almost to half for all of our test datasets compared to the acoustic i-vector system.

We also analyzed x-vector E-TDNN architecture (labeled as E-TDNN), which refers to the extended x-vector architecture described in Section 2.6.1. Based on results we can see significant improvement when using E-TDNN architecture over original x-vectors on all datasets. DET curve comparing both i-vector systems, baseline x-vector system and E-TDNN architecture on *sre18EvalCMN2* is shown in Figure 4.1. From DET curve we can see that both i-vector systems are very competitive especially for lower values of false acceptance ratio (around 1%), x-vector systems outperform i-vectors for all points of DET curve, and best results are achieved using E-TDNN architecture.

Based on this results and based on very recent publications, such as [47, 49, 33], we will focus on experimenting with x-vector based architectures since it outperforms i-vectors in accuracy and also in computational costs of forward-pass, which is a crucial factor for production usage and also allows simple usage of graphical computing units in many popular machine learning frameworks.

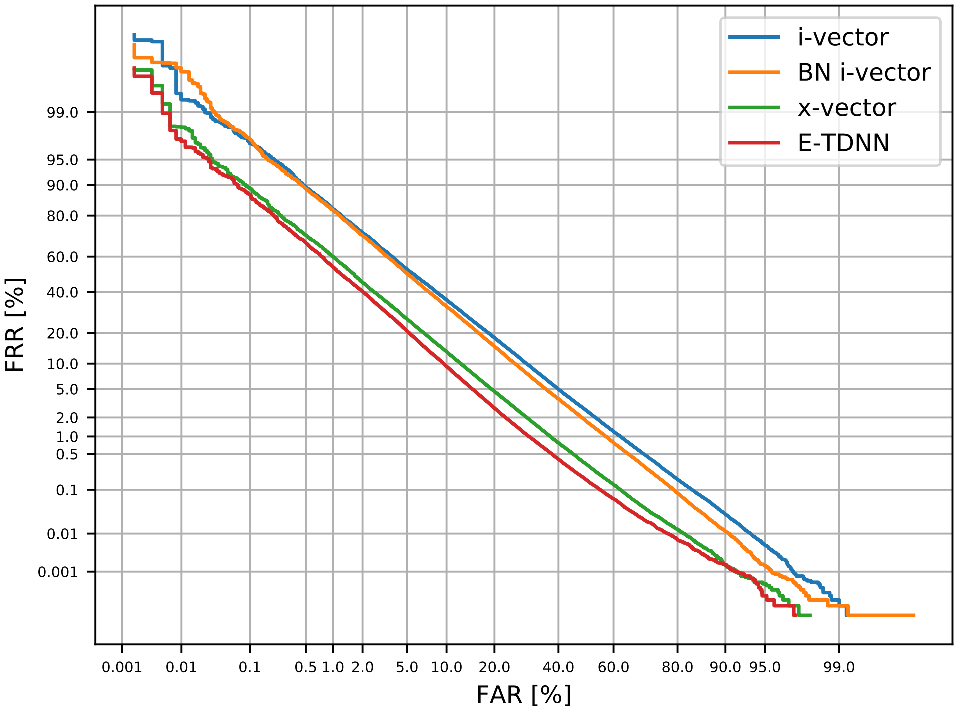

Figure 4.1: *Detection error tradeoff curve for baseline i-vector and x-vector based systems on sre18EvalCMN2 condition.*

4.2 Backend Experiments

As shown in Section 3.4, we can split our automatic system into two main components - embedding (i-vector or x-vector) extraction and backend. Here, we analyze different backends (PLDA models) and its impact on the performance and robustness of our systems. In our earlier experiments, we used PLDA backend described in Section 3.4.4, 512-dimensional x-vector projected into 128-dimensional space using LDA and scored using gaussian PLDA backend. We analyzed heavy-tailed PLDA described in Section 2.7.2, results are shown in Table 4.2. We can conclude, that Heavy Tailed PLDA backend yields very similar results to Gaussian PLDA with LDA dimensionality reduction on the *sre16EvalYUE* dataset, but shows outstanding results in terms of both equal error rate and $\mathrm{DCF}_{0.01}^{\mathrm{min}}$ on *sre18EvalCMN2*, where EER was reduced by 0.75% absolute. In our experiments in general, we could see better results with HT-PLDA backend and HT-PLDA is also very robust across domains, when values of CLLR across different evaluation conditions were lower compared to Gaussian PLDA. In our experiments, we use degrees of freedom equals to 2 and output dimensionality equals to 128, which are very close to values from [44].

4.2.1 Domain Adaptation

So far, we were not anyhow adapting our speaker recognition systems to evaluation conditions. In both NIST SRE16 and NIST SRE18 the part of development datasets also

Table 4.2: *Results using different PLDA backends - Gaussian PLDA (G-PLDA) backend with LDA dimensionality reduction and Heavy-Tailed PLDA (HT-PLDA) using E-TDNN system.*

System	Backend	sre16EvalYUE		sre18DevCMN2		sre18EvalCMN2	
		EER[%]	$\mathrm{DCF}_{0.01}^{\mathrm{min}}$	EER[%]	$\mathrm{DCF}_{0.01}^{\mathrm{min}}$	EER[%]	$\mathrm{DCF}_{0.01}^{\mathrm{min}}$
E-TDNN	G-PLDA + LDA	5.35	0.484	9.27	0.589	9.72	0.650
E-TDNN	HT-PLDA	5.17	0.478	8.77	0.578	8.97	0.628

consisted of *unlabeled* data, in case of NIST SRE18 these *unlabeled* data had assigned phone numbers; therefore this data might also be used for *supervised* adaptation in the same manner. We have taken advantage of this fact and used this data to adapt our system to target data. We ran experiments with three techniques for domain adaptation:

1. Mean normalization of speaker embedding from domain data. This technique should center speaker embeddings, so they have zero mean, which is expected by PLDA model.

2. Unsupervised and supervised adaptation of Heavy Tailed PLDA. Adaptation of PLDA enables the model to better match distribution in in-domain data.

3. Score normalization, we used adaptive s-norm as shown in Section 2.7.3 using the top 200 scores for computing statistics.

Naturally, all three mentioned techniques could be used at the same time, since they operate on different units. Our results are summarized in Table 4.3. Based on results, we can conclude that our adaptation techniques always improved results for *sre16EvalYUE* condition and for this condition, best results were achieved when using mean normalization, s-norm and unsupervised PLDA adaptation - EER 3.53% and $\mathrm{DCF}_{0.01}^{\mathrm{min}}$ 0.329, making 32% relative improvement for EER and 31% relative improvement for $\mathrm{DCF}_{0.01}^{\mathrm{min}}$over the system without adaptation. For *sre18EvalCMN2* condition, we noticed, that unsupervised adaptation of HT-PLDA had a negative impact on results in all cases, on the other hand, best results were obtained using mean normalization, s-norm and supervised adaptation of HT-PLDA - EER 7.06% and $\mathrm{DCF}_{0.01}^{\mathrm{min}}$ 0.539, making 21% relative improvement for EER and 14% improvement for $\mathrm{DCF}_{0.01}^{\mathrm{min}}$over the system without adaptation. DET curves for systems without adaptation and best systems with adaptation are shown in Figure 4.2 for *sre16EvalYUE* and in Figure 4.3 for *sre18EvalCMN2*, respectively.

4.3 Processing Speed

Processing speed is a critical factor in production systems, which directly allows end users to process more data while keeping the same computational costs. Also, smaller memory consumption and a smaller number of floating operations allows direct use in the Internet of Things (IoT) devices or mobile phones. This factor also influences training time, which in the state-of-the-art deep learning models usually extends days, weeks and sometimes even months. In this section, we include this factor besides the performance of our system and analyze multiple approaches to speeding up the computation. We will use faster than

Table 4.3: *Results using domain adaptation to evaluation conditions. The first column of table contains three letters corresponding to the system setup, first is usage of mean normalization (Y - used, N - not used), second part is s-norm (Y - used, N - not used) and last third is HT-PLDA adaptation (N - not used, U - unsupervised adaptation, S - supervised adaptation). Therefore, the first system (N/N/N - without any adaptation) is exactly the same as our HT-PLDA baseline. Since for NIST SRE16 there were no labels, we have not used the supervised adaptation of HT-PLDA.*

Adaptation	sre16EvalYUE		sre18DevCMN2		sre18EvalCMN2	
	EER[%]	$\mathrm{DCF}_{0.01}^{\mathrm{min}}$	EER[%]	$\mathrm{DCF}_{0.01}^{\mathrm{min}}$	EER[%]	$\mathrm{DCF}_{0.01}^{\mathrm{min}}$
N/N/N	5.17	0.478	8.77	0.578	8.97	0.628
Y/N/N	4.50	0.455	8.11	0.568	7.85	0.574
N/Y/N	4.28	0.345	7.45	0.469	7.50	0.517
Y/Y/N	4.00	0.336	7.16	0.480	7.28	0.524
N/N/U	4.29	0.433	9.34	0.682	10.05	0.691
N/N/S			7.89	0.578	8.96	0.636
Y/N/U	3.71	0.367	8.12	0.676	8.86	0.642
Y/N/S			6.93	0.569	7.58	0.548
Y/Y/U	3.53	0.329	7.54	0.550	7.61	0.550
Y/Y/S			6.60	0.507	7.06	0.539

real-time (FTRT) metric described as follows:

$$FTRT = \frac{time_{speech}}{time_{processing}} \tag{4.1}$$

where $time_{speech}$ refers to amount of speech declared using our VAD and $time_{processing}$ refers to amount of processing time. Especially deep models may benefit from usage of graphical processing units (GPUs), therefore we will report this metric using CPU and also GPU.

In all our experiments we used the same VAD and same input features. We will report only processing time and RAM consumption of forward pass using a single CPU core. For CPU we used python implementation in NumPy [1] with MKL backend [2] using AMD EPYC 7301 16-Core Processor. For GPU we used Theano [3] in python with GeForce GTX 1080 GPU. Our measurements were repeated for 5 times, and we used the mean of these values for reporting. We used 2798912 input frames from 425 different audio recordings.

There are multiple strategies on how to improve processing speed, such as:

1. Shrinking network size - the smaller architecture of NN, when we used 256 neurons in time-delay layers and dense layers and 750 neurons before pooling while keeping the same dimensionality of x-vector (denoted as *E-TDNN small*).

2. Skipping frames - we can skip each odd frame of the input so we will effectively use only half of the frames (denoted as $fs1$). Since we are using TDNN, we can also do this at any time-delay layer (denoted as $fs2_i$ where i refers to the index of the layer where we are performing frame skipping.

[1] http://www.numpy.org/

[2] https://software.intel.com/en-us/mkl

[3] http://deeplearning.net/software/theano/

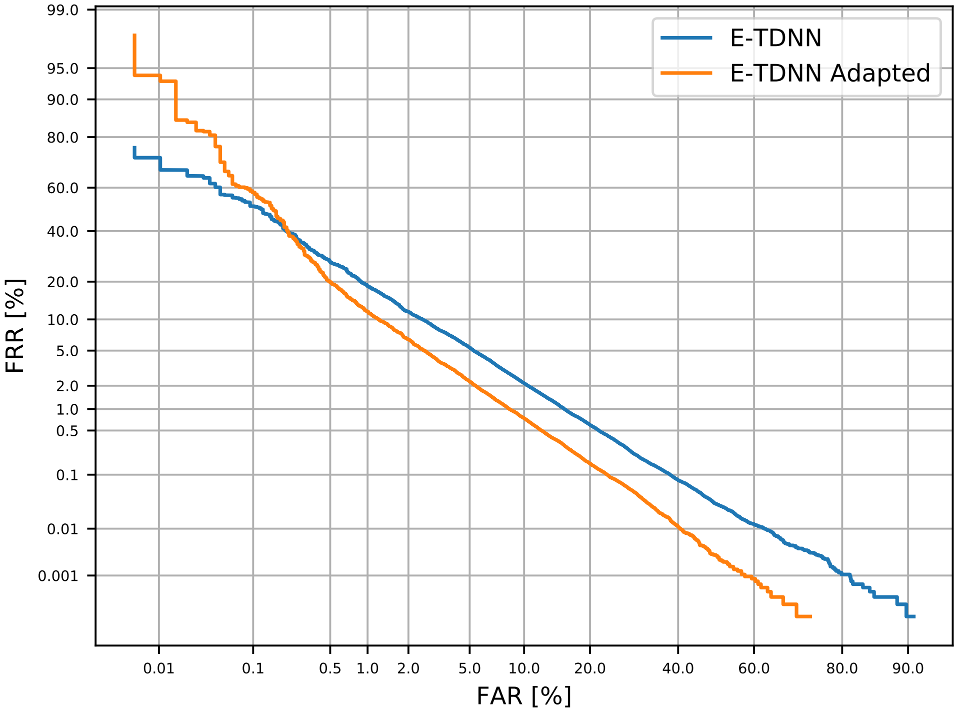

Figure 4.2: *Detection error tradeoff curve for E-TDNN system without adaptation and adapted system (Y/Y/U) on sre16EvalYUE condition.*

Based on results in Table 4.4, we can see, that both, shrinking network size and also skipping frames boosted computational speed significantly for both CPU and GPU, while achieving very good results compared to default full system. Especially E-TDNN $fs2_0$ system achieved almost same results as system, which is more than 2 times slower on CPU and 2.4 times slower on GPU. GPU processing time is 317 times faster than single CPU.

However, it is important to note, that embedding extraction is not the only part of the system pipeline described in Section 3.4. Based on our experiments, extraction of the MFCC features is very fast and takes only around 1% of the total time of the pipeline. VAD we used is based on neural networks and therefore can be easily accelerated on GPU and usually took around 3% of the total time of the pipeline. Based on this fact, we can expect, that the whole pipeline of E-TDNN $fs2_0$ system could achieve around 4500 FTRT on GPU.

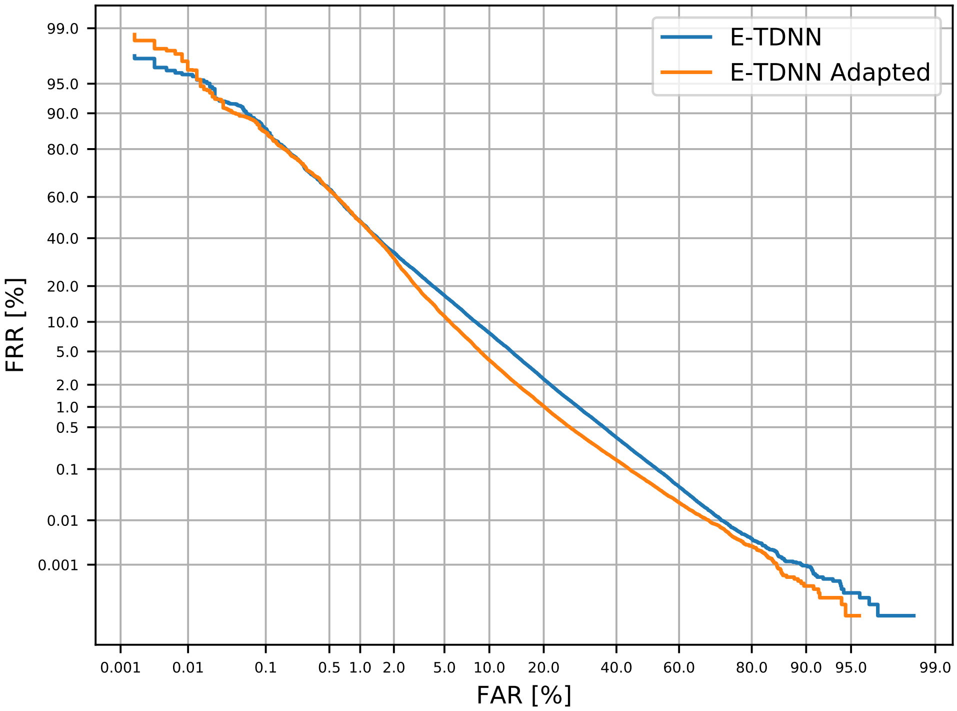

Figure 4.3: *Detection error tradeoff curve for E-TDNN system without adaptation and adapted system (Y/Y/S) on sre18EvalCMN2 condition.*

Table 4.4:

System	sre16EvalYUE EER[%]	sre18DevCMN2 EER[%]	sre18EvalCMN2 EER[%]	RAM GB	FTRT CPU	 GPU
E-TDNN	5.35	9.27	9.72	0.4	7.44	1985.04
E-TDNN small	5.59	10.03	10.35	0.3	11.45	2120.39
E-TDNN $fs1$	5.83	10.43	10.55	0.4	14.71	4664.85
E-TDNN $fs2_0$	5.35	10.03	10.05	0.4	14.96	4744

Table 5.1: *Baseline results on VAST-similar datasets for systems trained on 8 kHz mainly telephone data.*

System	sitwEvalC-C		voxc1	
	EER[%]	$\mathrm{DCF}_{0.01}^{\mathrm{min}}$	EER[%]	$\mathrm{DCF}_{0.01}^{\mathrm{min}}$
i-vector	13.37	0.791	16.40	0.928
BN i-vector	10.69	0.657	13.27	0.856
x-vector	7.16	0.559	9.00	0.676
E-TDNN	5.90	0.519	7.74	0.599

Chapter 5

Experiments - VAST condition

In this chapter, we analyze the performance of our systems on wideband conditions. First, we examine one to one trials on wideband evaluation sets. In the next part, we focus on multi-speaker recordings and diarization.

5.1 Baseline Systems

Baseline results for i-vector and x-vector system for VAST (wideband) conditions are shown in Table 5.1. Similarly to CMN2 baseline in Table 4.1, wideband datasets shows the same trend in terms of EER, bottleneck i-vectors slightly outperforms acoustic i-vectors, and x-vectors greatly outperform i-vector based architectures. E-TDNN again performs the best.

5.2 Domain Specific System

Here, we tried to adapt our system to target data during system training and therefore use only wideband data for system training. Since development corpus for SRE18 VAST condition is very small and not statistically reliable, it was not used for evaluation nor adaptation. For training we used VoxCeleb1 [35] and VoxCeleb2 [10] training sets, we trained extractor (x-vector NN) and also PLDA model on the same set.

We used the following modifications compared to original recipe [49] for all our experiments based on [33]:

- 9 epochs instead of 3 in the original recipe

- total 512 000 augmentations instead of 128 000 in the original recipe
- concatenate all utterances from a single session with one second of silence between every utterance.

Results for domain-specific systems are shown in Table 5.2. When we compare these results to results in Table 5.1, we can see that using domain-specific data is crucial for system's performance and even with our best E-TDNN system trained on telephone data with EER 5.90% on *sitwEvalC-C* we are not competitive with baseline x-vector system trained on wideband data with EER 4.89%.

In our experiments, we slightly changed the topology of TDNN to accept a larger context; these modifications are shown in Table 5.3 and are marked with suffix *LC (large context)*. We can conclude, that extending the context of TDNN improved results in terms of EER and also for another operating point. Also, we can see a very significant gain in using 16k sample rate over 8k sample rate - for competitive systems x-vector LC with 8k sample rate and 16k sample rate respectively; we can see almost 30% relative improvement in terms of EER.

We also trained our ETDNN without concatenating VoxCeleb audios, and these results are marked with suffix *cuts* and number before refers to number of augmentation that was used - it is important to note, that we used only 512 000 augmentations for NN in one case, which is less than half of the clean audios, therefore the system has not seen most of the data compared to previous cases, however results are competitive to baseline x-vector architecture but training requirements for computational resources are much lower. According to our experiments, original non-concatenated audios were not helpful for system training and therefore, we have not analyzed this scenario more.

In our experiments we also modified the process of arks creation (container with input features which are directly used for training). In default setup, training arks are created from 2-3 seconds long utterances randomly sampled from full utterance. We used following modifications to original Kaldi recipe:

```
sid/nnet3/xvector/get_egs.sh --cmd "$train_cmd_run_xv" \
    --nj 16 \
    --stage 0 \
    --frames-per-iter 100000000 \
    --frames-per-iter-diagnostic 100000 \
    --min-frames-per-chunk 200 \
    --max-frames-per-chunk 300 \
    --num-diagnostic-archives 3 \
    --num-repeats 15 \
    "$data" $egs_dir
```

These results are shown in Table 5.2 as system *E-TDNN arks* and actually yields best results for both testing conditions.

Also, we experimented with modifications of E-TDNN topology. Extending context in original x-vectors showed as crucial, and we extended context of original E-TDNN adding single TDNN-ReLU and Dense-ReLU layer with context $\{t-5, t, t+5\}$ after layer 8 from Table 2.2. This result is labeled as E-TDNN arks LC (large context). In our experiments we also stacked more time-delay layers on top of each other, creating vast network described in Table 5.4, extending context to 47 frames. This architecture is labeled as E-TDNN arks VLC (very large context). Based on results, we can conclude that extending context

Table 5.2: *Results for domain specific systems on VAST-similar datasets without using diarization.*

System	Sample Rate	sitwEvalC-C		voxc1	
		EER[%]	$\text{DCF}_{0.01}^{\text{min}}$	EER[%]	$\text{DCF}_{0.01}^{\text{min}}$
x-vector	8k	4.89	0.448	6.61	0.634
x-vector LC	8k	3.85	0.392	5.22	0.56
x-vector LC	16k	2.74	0.268	2.99	0.33
E-TDNN	16k	2.60	0.242	2.77	0.286
E-TDNN 500k cuts	16k	4.35	0.389	3.23	0.375
E-TDNN 5m cuts	16k	3.31	0.312	2.60	0.303
E-TDNN arks	16k	2.46	0.231	2.35	0.272
E-TDNN arks LC	16k	2.38	0.209	2.40	0.254
E-TDNN arks VLC	16k	2.49	0.237	2.26	0.251
E-TDNN arks LC HT-PLDA	16k	2.02	0.199	2.27	0.251

Table 5.3: *Configuration of TDNN for x-vector extraction using larger context. Bold values are our modifications of the original [49] architecture. X-vectors are extracted at layer segment6 before the nonlinearity.*

Layer	Layer context	Total context
frame1	[t-2,t+2]	5
frame2	{**t-4**, t-2,t,t+2,**t+4**}	13
frame3	{**t-6**,t-3,t,t+3,**t+6**}	19
frame4	{t}	19
frame5	{t}	19
stats pooling	[0, T]	T
segment6	{0}	T
segment7	{0}	T
softmax	{0}	T

is beneficial also for E-TDNN architecture, yielding 2.38% EER and 0.209 $\text{DCF}_{0.01}^{\text{min}}$ for E-TDNN arks LC system for *sitwEvalC-C*. E-TDNN arks VLC does not outperform other systems and considering the number of parameters in the network, which is 2.5 times bigger than in E-TDNN and therefore takes much longer time to train, we will not experiment with this architecture further.

As shown in previous experiments, HT-PLDA is shown as an excellent choice for the backend, and also for wideband systems we included it into our experiments. E-TDNN arks LC HT-PLDA refers to our so far best system with HT-PLDA backend and yields best results for *sitwEvalC-C*, 2.02% EER and 0.199 $\text{DCF}_{0.01}^{\text{min}}$.

DET curve for corresponding systems on *sitwEvalC-C* condition is in Figure 5.1. We can see a significant gain in using HT-PLDA backend over G-PLDA.

Table 5.4: *Extended TDNN x-vector architecture with very large context of 47 frames.*

Layer	Layer Type	Layer context	Size
1	TDNN-ReLU	[t-5,t+5]	512
2	Dense-ReLU	t	512
3	TDNN-ReLU	[t-4, t+4]	512
4	Dense-ReLU	t	512
5	TDNN-ReLU	[t-3, t+3]	512
6	Dense-ReLU	t	512
7	TDNN-ReLU	[t-2, t+2]	512
8	Dense-ReLU	t	512
9	TDNN-ReLU	{t-2, t, t+2}	512
10	Dense-ReLU	t	512
11	TDNN-ReLU	{t-3, t, t+3}	512
12	Dense-ReLU	t	512
13	TDNN-ReLU	{t-4, t, t+4}	512
14	Dense-ReLU	t	512
15	Dense-ReLU	t	512
16	Dense-ReLU	t	1500
17	Pooling (mean + stddev)	Full-seq	2x1500
18	Dense(Embedding)-ReLU		512
19	Dense-ReLU		512
20	Dense-SoftMax		512

5.3 Diarization in the Loop

In this section we analyze the performance of our system on testing conditions which necessarily does not contain single speaker at enroll or test side; therefore it should be sensible to run automatic diarization systems before performing speaker verification.

Suppose $R(,)$ is the PLDA log-likelihood ratio score, $\boldsymbol{u}$ is the x-vector for the enrolled speaker and $\boldsymbol{v_1}, \boldsymbol{v_2}, ..., \boldsymbol{v_N}$ are the x-vector for each of the N speakers in the test recording. To perform speaker recognition, log-likelihood is computed as follows:

$$R(enroll, test) = max\{R(\boldsymbol{u}, \boldsymbol{v_1}), ..., R(\boldsymbol{u}, \boldsymbol{v_N})\} \tag{5.1}$$

We analyze the performance of our best systems with and without diarization; results are shown in Table 5.5, for all our experiments we used the diarization system described in Section 3.4.5.

DET curve for *sre18EvalVAST* condition is shown in Figure 5.2. DET curves show us that there is a minimal difference between the x-vector LC system and E-TDNN, evaluation dataset is still very small and results may be noisy. We can conclude, that diarization helps for all our systems on *sitwEvalM-C* condition by 20% in terms of EER and also by 20% for $\text{DCF}_{0.01}^{\text{min}}$. On *sre18EvalVAST* condition, however, there is almost no gain in performance when using diarization. Also, there is no gain in performance when using enrollment annotations provided by NIST compared to taking whole audio on the enrollment side.

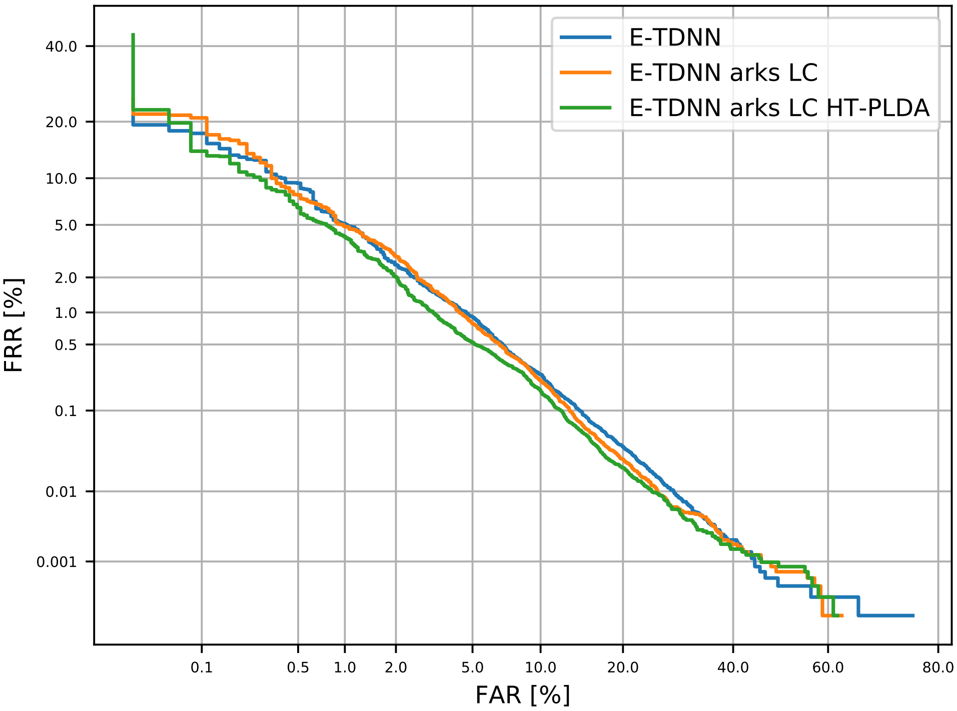

Figure 5.1: *Detection error tradeoff curve for systems trained on VoxCeleb1 and VoxCeleb2 data for sitwEvalC-C condition.*

Table 5.5: *Results for domain specific systems on VAST-similar datasets. Diarization column indicates whether diarization was used as pre-processing step. Enroll only means, that we used only enrollment segments annotated by NIST.*

System	Diarization	sitwEvalM-C		sre18EvalVAST	
		EER[%]	$\mathrm{DCF}_{0.01}^{\min}$	EER[%]	$\mathrm{DCF}_{0.01}^{\min}$
x-vector LC	no	5.20	0.363	13.33	0.746
E-TDNN	no	5.09	0.338	13.33	0.758
E-TDNN	enroll only			13.33	0.765
x-vector LC	yes	4.14	0.292	13.59	0.713
E-TDNN	yes	4.02	0.269	12.35	0.738
E-TDNN 5m cuts	yes	4.86	0.355	14.33	0.821

5.3.1 Speaker Diarization Implementation

In our work, we also implemented an automatic diarization system in python; it can be found together with one of our best x-vector models (E-TDNN) at [1]. The work on diarization is based on work done at 2017 Jelinek Summer Workshop on Speech and Language Technology

[1] https://github.com/Jamiroquai88/VBDiarization

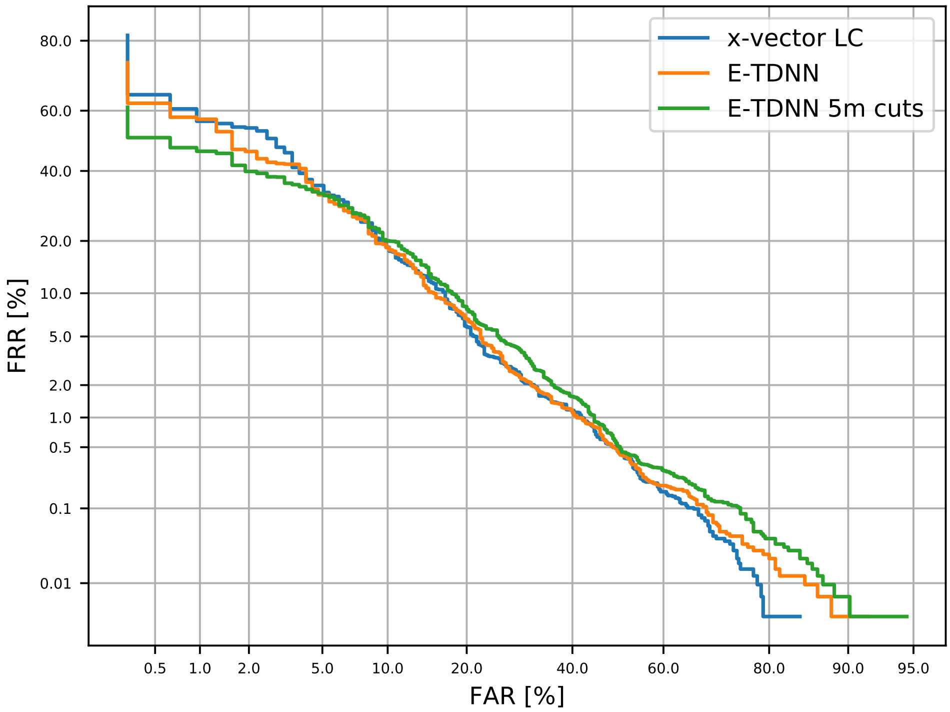

Figure 5.2: *Detection error tradeoff curve for systems trained on VoxCeleb1 and VoxCeleb2 data for sre18EvalVAST condition using diarization marks for test and oracle annotations for enrollment.*

(JSALT) at CMU [2]. The primary motivation to implement own speaker diarization system is to have the possibility to experiment with diarization sufficient for speaker verification described later.

In our implementation, we run clustering on top of x-vectors, which represent concise segments from one second up to two seconds in audio recording. At first, we analyzed the main topic of diarization - *who spoke when*. We used AMI corpus [3] for this purpose, using summed individual head-mounted microphones into a single channel. We used oracle voice activity detection generated from oracle rttm files which were also used for evaluation. In our approach, we clustered x-vectors using a k-means algorithm or in case of normalized x-vectors, using spherical k-means. In the second stage, we use PLDA scores in k-means clustering for fine-tuning of clustering. We used gaussian PLDA backend with LDA and l2 normalization. Since k-means expects the known value of k (number of clusters), we used the oracle number of speakers. We also analyzed the case, when the number of speakers is unknown and in this case, we used x-means algorithm [38] for estimating the number of clusters.

Results for these diarization scenarios are shown in Table 5.6. Based on the results, we can see that clustering using PLDA k-means reduced DER by 27% relative compared to

[2] https://www.lti.cs.cmu.edu/2017-jelinek-workshop
[3] http://groups.inf.ed.ac.uk/ami/corpus/

Table 5.6: *Diarization Error Rate (DER) results for evaluation and development part of AMI dataset containing summed individual head-mounted microphones in the single channel. We used a collar size of 250 ms.*

System	Clustering	DER [%]
E-TDNN	k-means	9.16%
E-TDNN	k-means + PLDA k-means	6.67%
E-TDNN	x-means + k-means + PLDA k-means	15.54%
E-TDNN	ahc	14.09%

spherical k-means only. We can conclude, that x-means algorithm is not the best choice for estimating the number of clusters and in our experiments usually estimated a smaller number of clusters than there were speakers. We also experimented with clustering using AHC, labeled as *ahc* in Table 5.6. In the case of AHC we trained a linear Gaussian model with two components with shared variance for calibration of scores. The threshold of the linear Gaussian model was used in AHC, and the number of speakers was estimated together with per-embedding labels. We can see, that AHC slightly outperforms x-means with PLDA k-means model by 1.45% DER.

The goal of these experiments was to show, that speaker diarization is working quite well in terms of DER and the best system with PLDA k-means is investigated further in multi-speakers scenario focused on speaker recognition.

5.3.2 Diarization Sufficient for Speaker Verification

It is important to note, that diarization task of defining *who spoke when* is not necessarily the same task to the one in speaker verification when we want to know the answer to the question *is this enrollment speaker in that test recording, which may include multiple speakers*? This could be the case, if diarization itself yielded outstanding results on any testing conditions, which is not the case, as shown in recent DIHARD challenge [4] where using ground truth VAD the best systems got 23.73% DER [43] and without these ground truth labels achieving much worse performance, only 35.53% DER [15]. For some of the domains, it was even better to say that there is a single speaker in the whole recording, even when it was not a case.

Therefore, as shown in recent publication [48], where authors show excellent results on multi-speaker conversations, we analyzed speaker diarization specific for speaker recognition.

AHC-based diarization typically requires a well-chosen cluster stopping threshold to achieve good performance. This threshold is sensitive to the domain of the data, and a poorly chosen threshold will result in bad performance. In K-Means algorithm it is a similar case with a number of clusters, which is frequently unknown. This is a particularly concerning possibility when a reliable development set is not available, as in case of NIST SRE 2018 VAST data. To improve robustness, the authors of [48] propose a simple change in the clustering algorithm. Instead of relying on a tuned AHC threshold, they estimate the maximum number of speakers in recording and run clustering iteratively with the different number of clusters exactly K times, with $k \in \{1, 2, ..., K\}$. The final number of cluster is

[4] https://coml.lscp.ens.fr/dihard/2018/results.php

Table 5.7:

System	Diarization	sitwEvalM-C		sre18EvalVAST	
		EER[%]	$\mathrm{DCF}^{\mathrm{min}}_{0.01}$	EER[%]	$\mathrm{DCF}^{\mathrm{min}}_{0.01}$
E-TDNN	ahc+vb	4.02	0.269	12.35	0.738
E-TDNN	iterative k-means	2.87	0.262	12.03	0.789

then defined as $N = \frac{K(K+1}{2}$, where K is a maximal number of speakers in the recording. The output of clustering is then N cluster centers.

We tried to replicate this setup with $K = 5$ as shown in the original paper, but with the K-Means algorithm instead of AHC. Our results are shown in Table 5.7, we compared our previous results from multi-speaker conversations (ahc+vb) with our new approach (iterative k-means). Based on results, we can conclude that for *sitwEvalM-C* there is a significant gain in terms of EER, reduced from 4.02% to 2.87%. Our approach was less successful for the *sre18EvalVAST* condition, where we can see a small improvement in EER, but system obtains worse results for $\mathrm{DCF}^{\mathrm{min}}_{0.01}$. When we compare these results to results of E-TDNN system on *sitwEvalC-C* condition with EER 2.60% and $\mathrm{DCF}^{\mathrm{min}}_{0.01}$ 0.242, a scenario with a single speaker at the test side is still more successful. However, the difference is substantially smaller. For *sre18EvalVAST* condition, it is however not clear, why this approach does not improve performance.

Chapter 6

Conclusion

6.1 Experiments Summary

In this experimental work, we analyzed the state-of-the-art speaker verification pipeline using x-vector based speaker embeddings which superseded i-vectors in recent years. Developing a robust system which would work across various conditions, such as unseen language, distinctive acoustic conditions or problem of multi-speaker recordings remains very difficult.

We compared systems based on i-vectors to x-vectors on narrowband and wideband conditions. We could see, that data augmentation is an easily implemented and effective strategy for improving x-vectors performance. We showed, that using in-domain wideband data for training, in this case, VoxCeleb1 and VoxCeleb2, we were able to outperform systems trained on 8 kHz telephone data. VoxCeleb1 and VoxCeleb2 datasets are also extensive, containing over 1 million utterances from thousands of speakers and allow us to use state-of-the-art deep learning methods.

Score normalization and system adaptation, such as mean normalization using in-domain data and supervised and unsupervised adaptation of PLDA backend showed as crucial in tuning the system for specific evaluation conditions and led to improved results on *sre16EvalYUE* by 32% and on *sre18EvalCMN2* by 21% relative in terms of EER.

We also experimented with improving our scoring backend, and we used Heavy Tailed PLDA for scoring, yielding 2.02% EER on the *sitwEvalC-C* dataset, using an out-of-the-box system, without any adaptation to SITW dataset. Comparing our results on *voxc1* test dataset to ResNet architecture from [10], in terms of equal error rate using E-TDNN with HT-PLDA backend we obtained 2.27% EER compared to their 3.95%.

Also, our best wideband system produced during NIST SRE 2018 evaluations was used as one of the submission systems and was very competitive considering all submissions of other teams. Using diarization in speaker verification, however, still looks like a problematic area with very high error rates and should also be included as an active area of speech technology research. Interestingly, results for *sre18EvalVAST* condition remains very noisy and improvements showed for the *sitwEvalC-C* condition does not generalize.

6.2 Future Work

Our future work will be focused on experimenting more with E-TDNN architecture, such as extending the context of time-delay layers and stacking more of these layers into a network.

Also, we would like to experiment more with types of DNN architectures which show very results in face recognition, such as residual networks from [56]. Even though Kaldi toolkit is very easy to use and yields very good results, we would like to experiment more with popular python toolkits, such as TensorFlow or PyTorch and try to replicate results obtained in Kaldi.

Also, we want to focus more on diarization, either for scenario *who spoke when*, when our results could be improved using variational Bayes as an additional step and the same assumption might be applied for diarization scenario focused on speaker verification. We would like to release our diarization code and models for the research community to use.

Bibliography

[1] Abadi, M.; Agarwal, A.; Barham, P.; et al.: TensorFlow: Large-Scale Machine Learning on Heterogeneous Systems. 2015. software available from tensorflow.org. Retrieved from: http://tensorflow.org/

[2] Abbas, O. A.: Comparisons Between Data Clustering Algorithms. *International Arab Journal of Information Technology (IAJIT)*. vol. 5, no. 3. 2008.

[3] Agrawal, P.; Kapoor, R.; Agrawal, S.: A hybrid partial fingerprint matching algorithm for estimation of Equal Error Rate. In *Advanced Communication Control and Computing Technologies (ICACCCT), 2014 International Conference on*. IEEE. 2014. pp. 1295–1299.

[4] Auckenthaler, R.; Carey, M.; Lloyd-Thomas, H.: Score normalization for text-independent speaker verification systems. *Digital Signal Processing*. vol. 10, no. 1-3. 2000: pp. 42–54.

[5] Bishop, C. M.: *Pattern recognition and machine learning*. springer. 2006.

[6] Brümmer, N.; Du Preez, J.: Application-independent evaluation of speaker detection. *Computer Speech & Language*. vol. 20, no. 2-3. 2006: pp. 230–275.

[7] Brümmer, N.; Silnova, A.; Burget, L.; et al.: Gaussian meta-embeddings for efficient scoring of a heavy-tailed PLDA model. *arXiv preprint arXiv:1802.09777*. 2018.

[8] Burget, L.; Plchot, O.; Cumani, S.; et al.: Discriminatively trained probabilistic linear discriminant analysis for speaker verification. In *Acoustics, Speech and Signal Processing (ICASSP), 2011 IEEE International Conference on*. IEEE. 2011. pp. 4832–4835.

[9] Castan, D.; McLaren, M.; Ferrer, L.; et al.: Improving Robustness of Speaker Recognition to New Conditions Using Unlabeled Data. In *INTERSPEECH*. 2017. pp. 3737–3741.

[10] Chung, J. S.; Nagrani, A.; Zisserman, A.: VoxCeleb2: Deep Speaker Recognition. *arXiv preprint arXiv:1806.05622*. 2018.

[11] Clouse, D. S.; Giles, C. L.; Horne, B. G.; et al.: Time-delay neural networks: Representation and induction of finite-state machines. *IEEE Transactions on Neural Networks*. vol. 8, no. 5. 1997: pp. 1065–1070.

[12] Dahl, G. E.; Sainath, T. N.; Hinton, G. E.: Improving deep neural networks for LVCSR using rectified linear units and dropout. In *2013 IEEE international conference on acoustics, speech and signal processing*. IEEE. 2013. pp. 8609–8613.

[13] Dehak, N.; Kenny, P. J.; Dehak, R.; et al.: Front-end factor analysis for speaker verification. *IEEE Transactions on Audio, Speech, and Language Processing.* vol. 19, no. 4. 2011: pp. 788–798.

[14] Diez, M.; Burget, L.; Matějka, P.: Speaker Diarization based on Bayesian HMM with Eigenvoice Priors. In *Odyssey 2018, The Speaker and Language Recognition Workshop.* 2018.

[15] Diez, M.; Landini, F.; Burget, L.; et al.: BUT System for DIHARD Speech Diarization Challenge 2018. In *Proc. Interspeech.* 2018. pp. 2798–2802.

[16] Doddington, G.: Speaker recognition based on idiolectal differences between speakers. In *Seventh European Conference on Speech Communication and Technology.* 2001.

[17] Fér, R.; Matějka, P.; Grézl, F.; et al.: Multilingual bottleneck features for language recognition. In *Sixteenth Annual Conference of the International Speech Communication Association.* 2015.

[18] Ghahremani, P.; BabaAli, B.; Povey, D.; et al.: A pitch extraction algorithm tuned for automatic speech recognition. In *Acoustics, Speech and Signal Processing (ICASSP), 2014 IEEE International Conference on.* IEEE. 2014. pp. 2494–2498.

[19] Glembek, O.: *Optimalization of Gaussian Mixture Subspace Models and Related Scoring Algorithms in Speaker Verification.* PhD dissertation. Brno University of Technology, Faculty of Information Technology, Brno. 2012.

[20] Glembek, O.; Ma, J.; Matějka, P.; et al.: Domain adaptation via within-class covariance correction in i-vector based speaker recognition systems. In *Acoustics, Speech and Signal Processing (ICASSP), 2014 IEEE International Conference on.* IEEE. 2014. pp. 4032–4036.

[21] Hannun, A.; Case, C.; Casper, J.; et al.: Deep speech: Scaling up end-to-end speech recognition. *arXiv preprint arXiv:1412.5567.* 2014.

[22] Hinton, G.; Deng, L.; Yu, D.; et al.: Deep neural networks for acoustic modeling in speech recognition. *IEEE Signal processing magazine.* vol. 29. 2012.

[23] Kenny, P.: Bayesian speaker verification with heavy-tailed priors. In *Odyssey.* 2010. page 14.

[24] Kenny, P.; Boulianne, G.; Ouellet, P.; et al.: Joint factor analysis versus eigenchannels in speaker recognition. *IEEE Transactions on Audio, Speech, and Language Processing.* vol. 15, no. 4. 2007: pp. 1435–1447.

[25] Kinnunen, T.; Li, H.: An overview of text-independent speaker recognition: From features to supervectors. *Speech communication.* vol. 52, no. 1. 2010: pp. 12–40.

[26] Lukáš Burget, IKR Slides: Gaussian distribution. `https://www.fit.vutbr.cz/study/courses/IKR/public/prednasky/02_bayesovska_teorie/bayesovska_teorie.pdf`.

[27] Martin, A.; Doddington, G.; Kamm, T.; et al.: The DET curve in assessment of detection task performance. Technical report. National Inst of Standards and Technology Gaithersburg MD. 1997.

[28] Matějka, P.; Glembek, O.; Castaldo, F.; et al.: Full-covariance UBM and heavy-tailed PLDA in i-vector speaker verification. In *2011 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*. IEEE. 2011. pp. 4828–4831.

[29] Matějka, P.; Glembek, O.; Novotný, O.; et al.: Analysis of DNN approaches to speaker identification. In *Acoustics, Speech and Signal Processing (ICASSP), 2016 IEEE International Conference on*. IEEE. 2016. pp. 5100–5104.

[30] Matějka, P.; Novotný, O.; Plchot, O.; et al.: Analysis of Score Normalization in Multilingual Speaker Recognition. In *INTERSPEECH*. 2017. pp. 1567–1571.

[31] Matějka, P.; Plchot, O.; Novotný, O.; et al.: BUT-PT System Description for NIST LRE 2017.

[32] Matějka, P.; Zhang, L.; Ng, T.; et al.: Neural network bottleneck features for language identification. *Proc. IEEE Odyssey*. 2014: pp. 299–304.

[33] McLaren, M.; Castan, D.; Nandwana, M. K.; et al.: How to train your speaker embeddings extractor. In *Odyssey: The Speaker and Language Recognition Workshop, Les Sables d'Olonne*. 2018.

[34] McLaren, M.; Ferrer, L.; Castan, D.; et al.: The 2016 Speakers in the Wild Speaker Recognition Evaluation. In *INTERSPEECH*. 2016. pp. 823–827.

[35] Nagrani, A.; Chung, J. S.; Zisserman, A.: Voxceleb: a large-scale speaker identification dataset. *arXiv preprint arXiv:1706.08612*. 2017.

[36] Paszke, A.; Gross, S.; Chintala, S.; et al.: Automatic differentiation in PyTorch. 2017.

[37] Peddinti, V.; Povey, D.; Khudanpur, S.: A time delay neural network architecture for efficient modeling of long temporal contexts. In *Sixteenth Annual Conference of the International Speech Communication Association*. 2015.

[38] Pelleg, D.; Moore, A. W.; et al.: X-means: extending k-means with efficient estimation of the number of clusters. In *Icml*, vol. 1. 2000. pp. 727–734.

[39] Povey, D.; Ghoshal, A.; Boulianne, G.; et al.: The Kaldi Speech Recognition Toolkit. In *IEEE 2011 Workshop on Automatic Speech Recognition and Understanding*. IEEE Signal Processing Society. December 2011. iEEE Catalog No.: CFP11SRW-USB.

[40] Prince, S. J.; Elder, J. H.: Probabilistic linear discriminant analysis for inferences about identity. In *2007 IEEE 11th International Conference on Computer Vision*. IEEE. 2007. pp. 1–8.

[41] Reynolds, D. A.; Quatieri, T. F.; Dunn, R. B.: Speaker verification using adapted Gaussian mixture models. *Digital signal processing*. vol. 10, no. 1-3. 2000: pp. 19–41.

[42] Sadjadi, S. O.; Kheyrkhah, T.; Tong, A.; et al.: The 2016 NIST Speaker Recognition Evaluation. In *Interspeech*. 2017. pp. 1353–1357.

[43] Sell, G.; Snyder, D.; McCree, A.; et al.: Diarization is hard: Some experiences and lessons learned for the JHU team in the inaugural DIHARD challenge. In *Proc. Interspeech*. 2018. pp. 2808–2812.

[44] Silnova, A.; Brümmer, N.; Garcia-Romero, D.; et al.: Fast variational Bayes for heavy-tailed PLDA applied to i-vectors and x-vectors. *arXiv preprint arXiv:1803.09153*. 2018.

[45] Snyder, D.; Garcia-Romero, D.; McCree, A.; et al.: Spoken language recognition using x-vectors. In *Proc. Odyssey 2018 The Speaker and Language Recognition Workshop*. 2018. pp. 105–111.

[46] Snyder, D.; Garcia-Romero, D.; Povey, D.: Time delay deep neural network-based universal background models for speaker recognition. In *2015 IEEE Workshop on Automatic Speech Recognition and Understanding (ASRU)*. IEEE. 2015. pp. 92–97.

[47] Snyder, D.; Garcia-Romero, D.; Povey, D.; et al.: Deep neural network embeddings for text-independent speaker verification. In *Proc. Interspeech*. 2017. pp. 999–1003.

[48] Snyder, D.; Garcia-Romero, D.; Sell, G.; et al.: SPEAKER RECOGNITION FOR MULTI-SPEAKER CONVERSATIONS USING X-VECTORS.

[49] Snyder, D.; Garcia-Romero, D.; Sell, G.; et al.: X-vectors: Robust DNN embeddings for speaker recognition. *Submitted to ICASSP*. 2018.

[50] Snyder, D.; Ghahremani, P.; Povey, D.; et al.: Deep neural network-based speaker embeddings for end-to-end speaker verification. In *Spoken Language Technology Workshop (SLT), 2016 IEEE*. IEEE. 2016. pp. 165–170.

[51] Tranter, S. E.; Reynolds, D. A.: An overview of automatic speaker diarization systems. *IEEE Transactions on audio, speech, and language processing*. vol. 14, no. 5. 2006: pp. 1557–1565.

[52] Veselý, K.; Karafiát, M.; Grézl, F.: Convolutive bottleneck network features for LVCSR. In *2011 IEEE Workshop on Automatic Speech Recognition & Understanding*. IEEE. 2011. pp. 42–47.

[53] Veselý, K.; Karafiát, M.; Grézl, F.; et al.: The language-independent bottleneck features. In *2012 IEEE Spoken Language Technology Workshop (SLT)*. IEEE. 2012. pp. 336–341.

[54] Widrow, B.; Lehr, M. A.: 30 years of adaptive neural networks: perceptron, madaline, and backpropagation. *Proceedings of the IEEE*. vol. 78, no. 9. 1990: pp. 1415–1442.

[55] Wooters, C.; Huijbregts, M.: The ICSI RT07s speaker diarization system. *Multimodal Technologies for Perception of Humans*. 2008: pp. 509–519.

[56] Xie, W.; Nagrani, A.; Chung, J. S.; et al.: Utterance-level Aggregation For Speaker Recognition In The Wild. *arXiv preprint arXiv:1902.10107*. 2019.

Appendix A

Content of the CD

- **DP.pdf** - this document in *pdf* format
- **VBDiarization/** - diarization code from `https://github.com/Jamiroquai88/VBDiarization`
  - **configs/** - directory with configuration files
  - **examples/** - directory with examples, see mainly *diarization.py*
  - **models/** - directory containing pre-trained models
  - **vbdiar/** - code of the library
  - **LICENCE** - file with MIT licence
  - **README.md** - main README with steps how to install this package
  - **requirements.txt** - *python* requirements file
  - **setup.py** - setup script
- **pysid/** - python library for speaker verification
- **evaluator/** - configuration files and scripts for evaluation of datasets